Before Starting:

Websites and their usernames, passwords, and token

***PARABANK:

ParaBank | Administration (parasoft.com)

Username **john**

Password **demo**

***WEATHER:

Weather API - OpenWeatherMap

API KEY appid `926157617b49a877ea7e848f022353a9`

****URL: gorest

https://gorest.co.in/public/v2/users

https://gorest.co.in/public/v1/users

***API KEY :

Bearer 3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7

***Mealb

http://dev-mb.yoll.io/Account/Login

*****ToKEN API:

username: ElshanRasul

password: Elshan123

http://dev-mb.yoll.io/api/tokenauth/authenticate

***Params

Path Params starts with a **slash** : **/**448465465

Query Params starts with a **interrogative** mark: **?**id=1651684 ; we can see it when we put Params in (Key/Value)

***WSDL**

http://www.dneonline.com/calculator.asmx?WSDL

https://www.swi-prolog.org/pack/file_details/wsdl/examples/country.wsdl

***SWAGGER**

https://parabank.parasoft.com/

***login postman:**

john

demo

***Basic Auth**

APITeacher

12345

***Beautify XML Code**

Ctrl + Shift +F = to Beautify XML code

****Create Main Method**

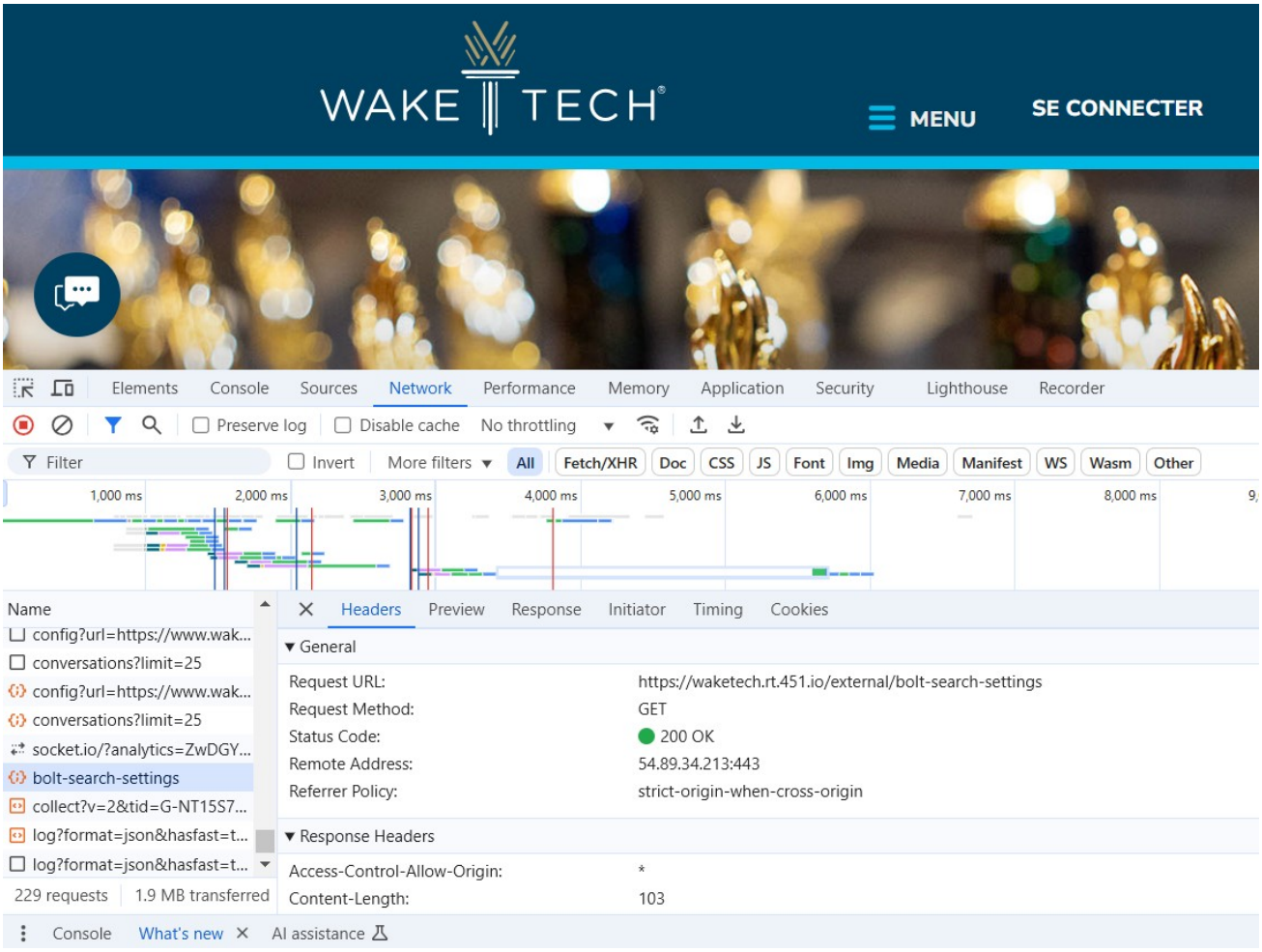
write main / Ctrl / Space

DURING INTERVIEW say we use only GET and Post

****You can see different Methods (Get, post…) on a Website by :

Click on Inspection of the Page/ Network/ Click on a link From Name Tag o nthe left:

Testers and DevOps need API documentation to start working


WAKE TECH
MENU
SE CONNECTER
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Wasm
Other
1,000 ms
2,000 ms
3,000 ms
4,000 ms
5,000 ms
6,000 ms
7,000 ms
8,000 ms
Name
config?url=https://www.wak...
conversations?limit=25
config?url=https://www.wak...
conversations?limit=25
socket.io/?analytics=ZwDGY...
bolt-search-settings
collect?v=2&tid=G-NT15S7...
log?format=json&hasfast=t...
log?format=json&hasfast=t...
229 requests
1.9 MB transferred
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL:
https://waketech.rt.451.io/external/bolt-search-settings
Request Method:
GET
Status Code:
200 OK
Remote Address:
54.89.34.213:443
Referrer Policy:
strict-origin-when-cross-origin
Response Headers
Access-Control-Allow-Origin:
*
Content-Length:
103
Console
What's new
AI assistance

Postman

AUTHENTICATION

Authentication checks your **identity** to confirm you are the same person who you said There are 3 types of authentication:

- **HTTP Basic Auth:** User provides a **username** and **password** in the **header** of request. In other word access token sent as the username
- **API Key**: a **unique generated value** is assigned to user.
  - This value can be generated by system. It may expires after certain time (ex: a year or 6 months). That value passed in header of request.
- **OAuth 2 Token :**
  - You get token created by Authorization server.
  - Its sent to API server using HTTP Bearer Tokens
  - It returns you a token (combination of letters,numbers and symbols). That token added to header of request. System identify your **authentication** and **authorization** based on that token.

Authentication vs. Authorization in Postman

1. **Authentication**: Verifies *who you are* (identity verification).
   Example: Logging in with a username and password.
2. **Authorization**: Determines *what you can do* (permissions).
   Example: Accessing admin-only resources after authentication.

Types of Authentication in Postman

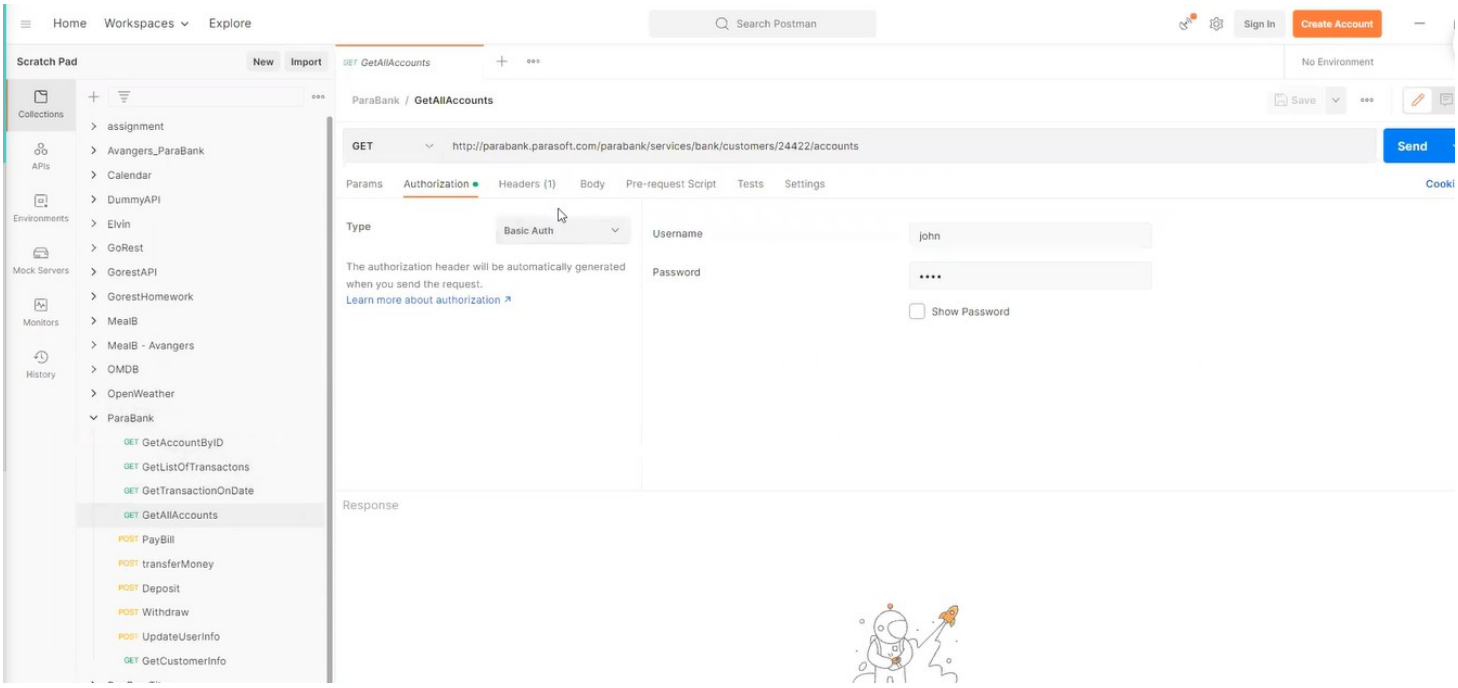
1. Basic Authentication

- Involves a **username** and **password** encoded in Base64.
- Common for simple, non-sensitive APIs.

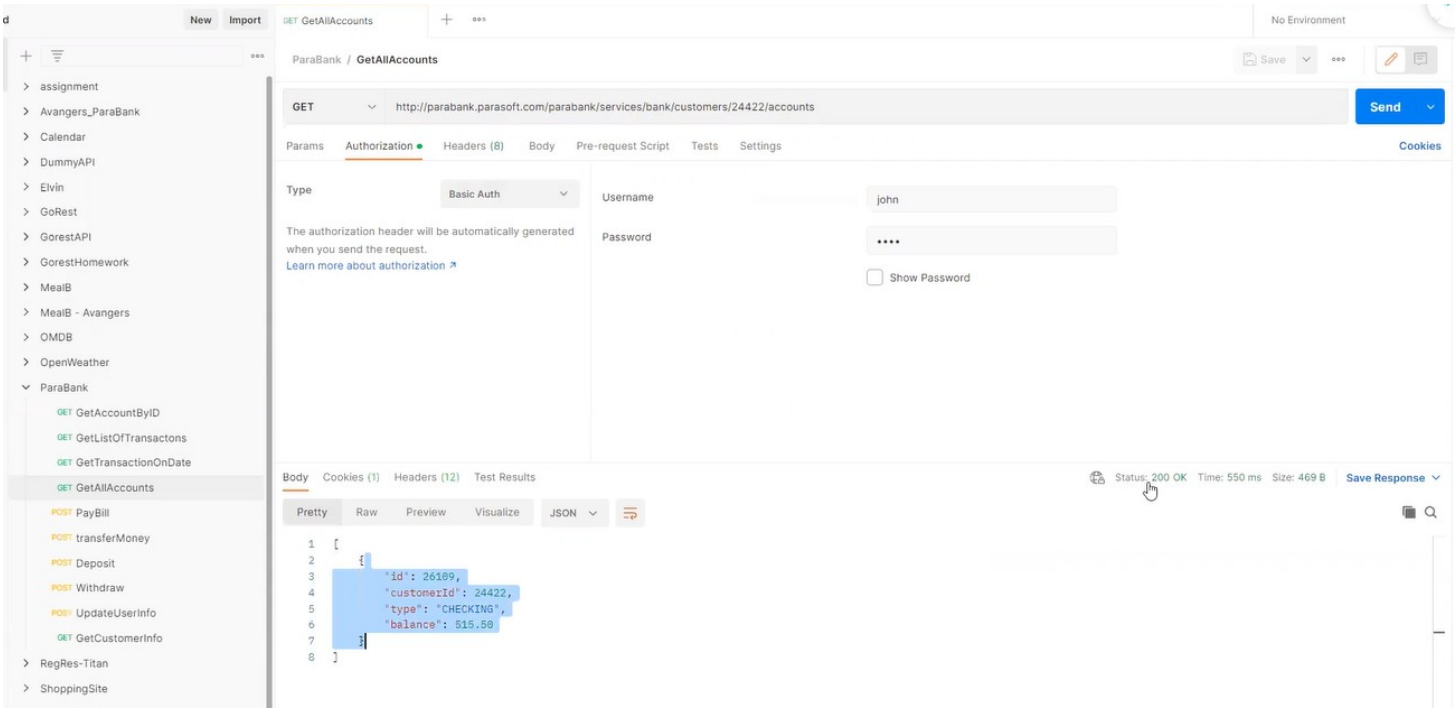
Example: Accessing user details in a ride-share app.

In Postman:

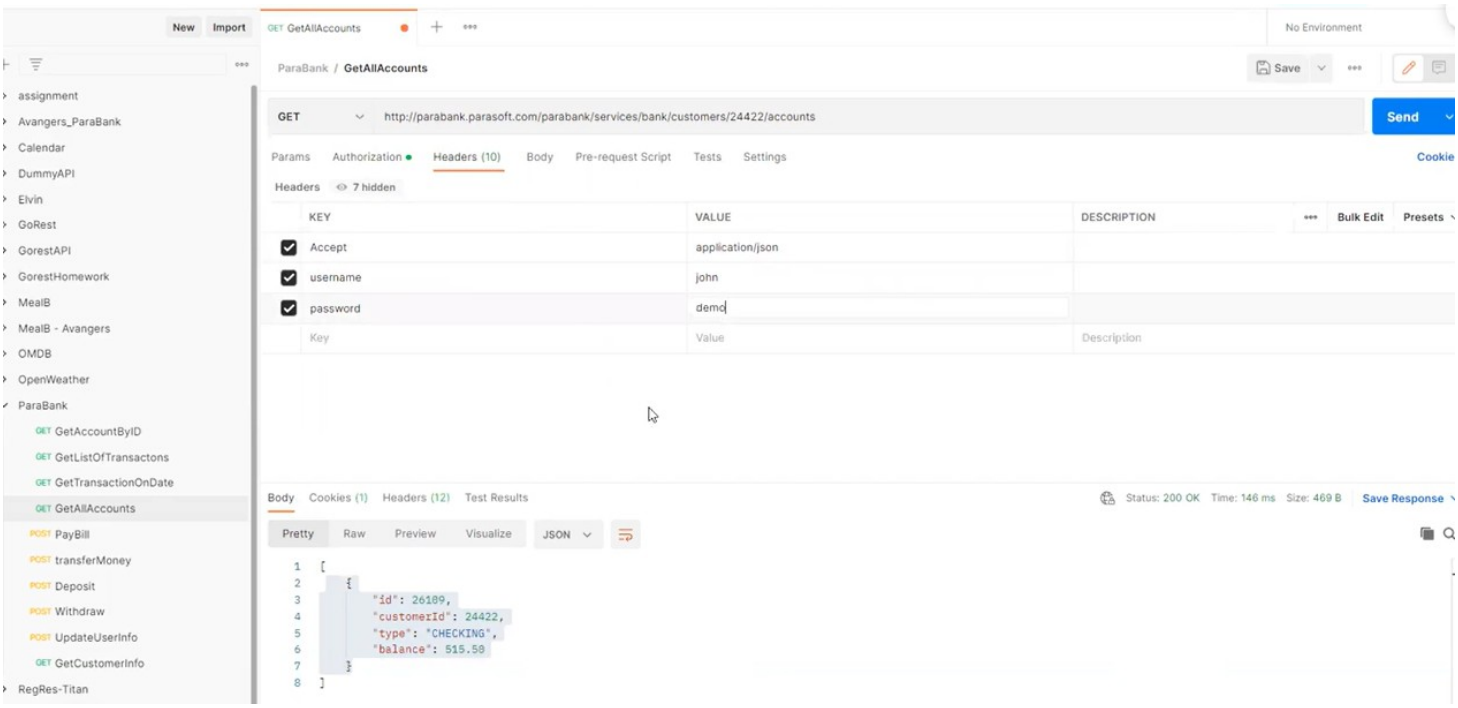
- **Go to the Authorization tab.**
- Select **Basic Auth**.
- Enter **Username**: john
- Enter **Password**: password123



When you submit the request, you get the response:

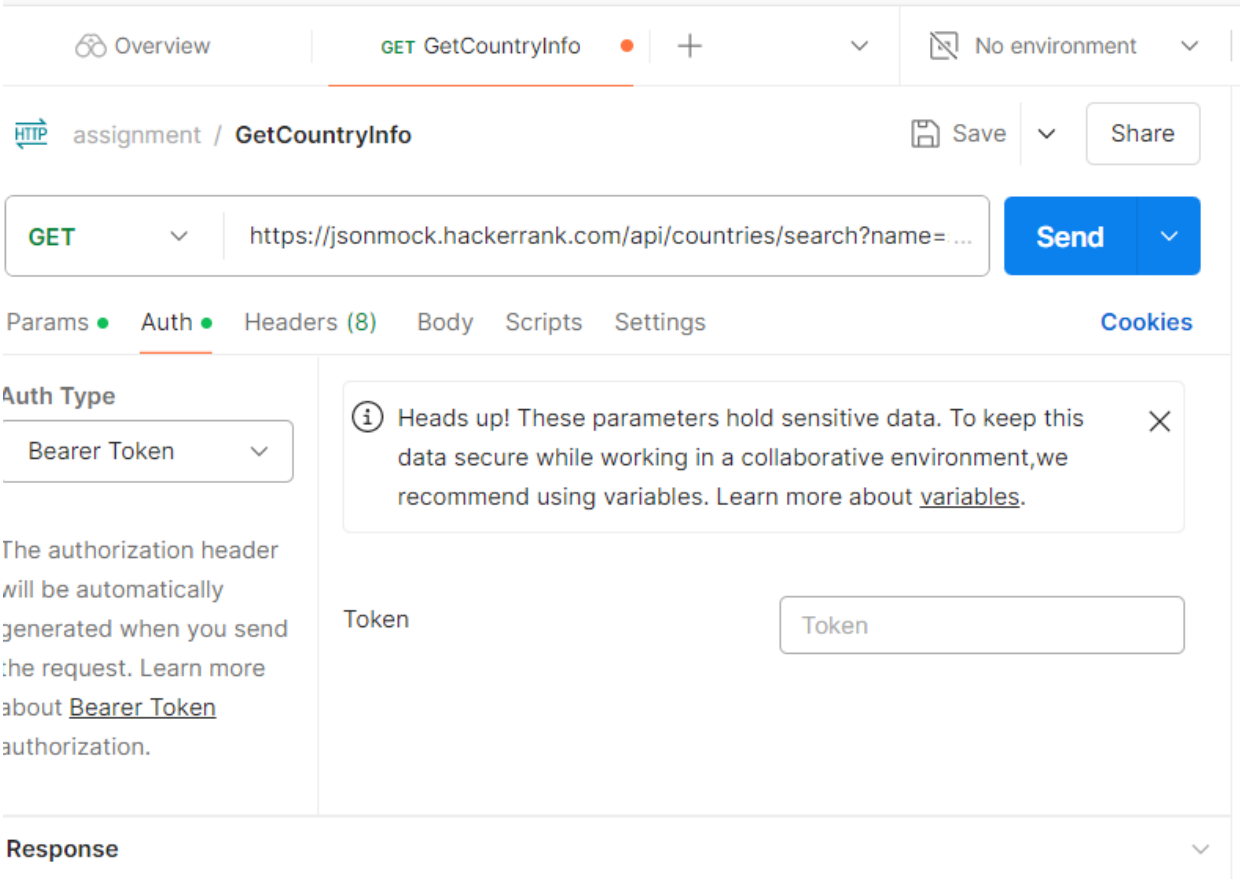


- OR you can go to the **Headers** tab, then put the username and password:



2. Bearer Token (Token-based Authentication)

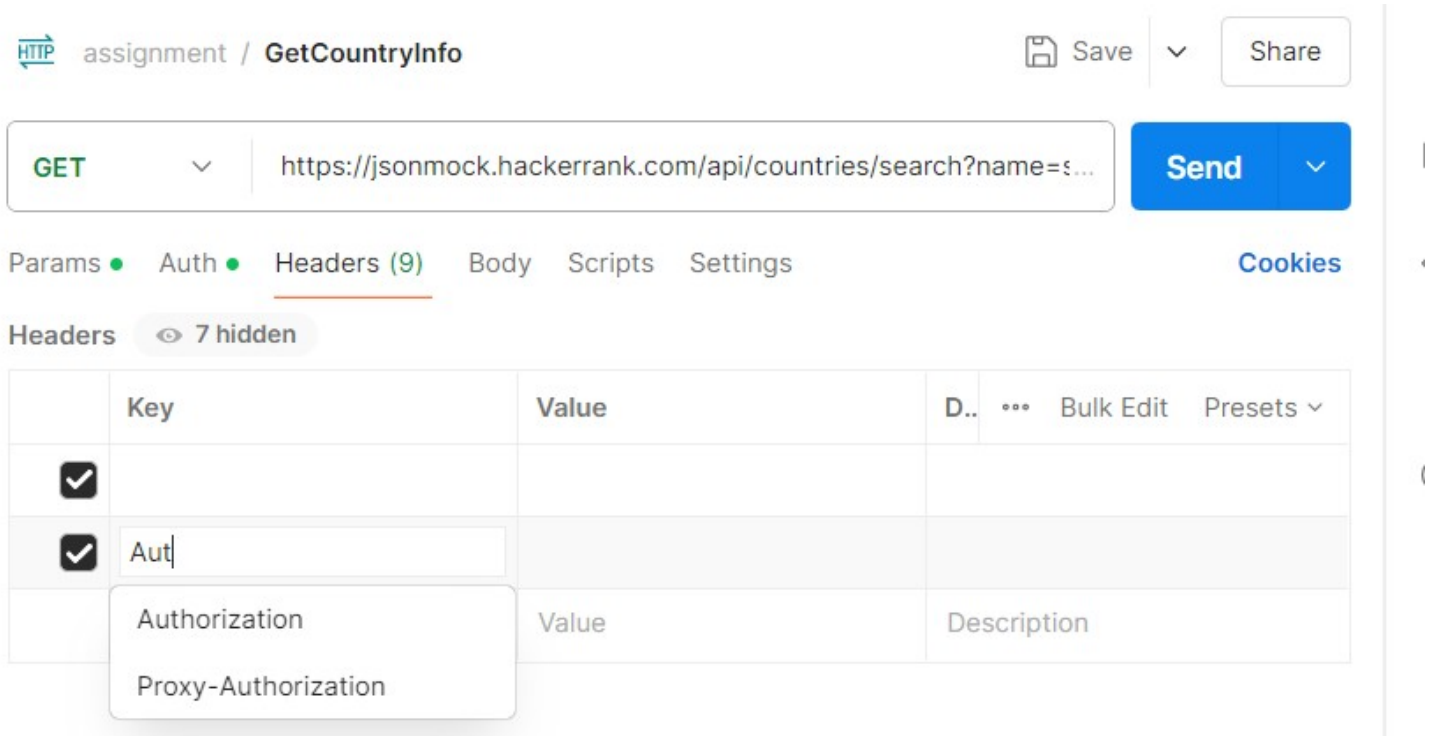
- Uses a token (e.g., JWT) for secure communication.
- Safer than basic authentication.
- You can use the **Bearer Token** in both the **Authorization** tab and the **Headers** tab in **Postman**



Go to the **Headers** tab.

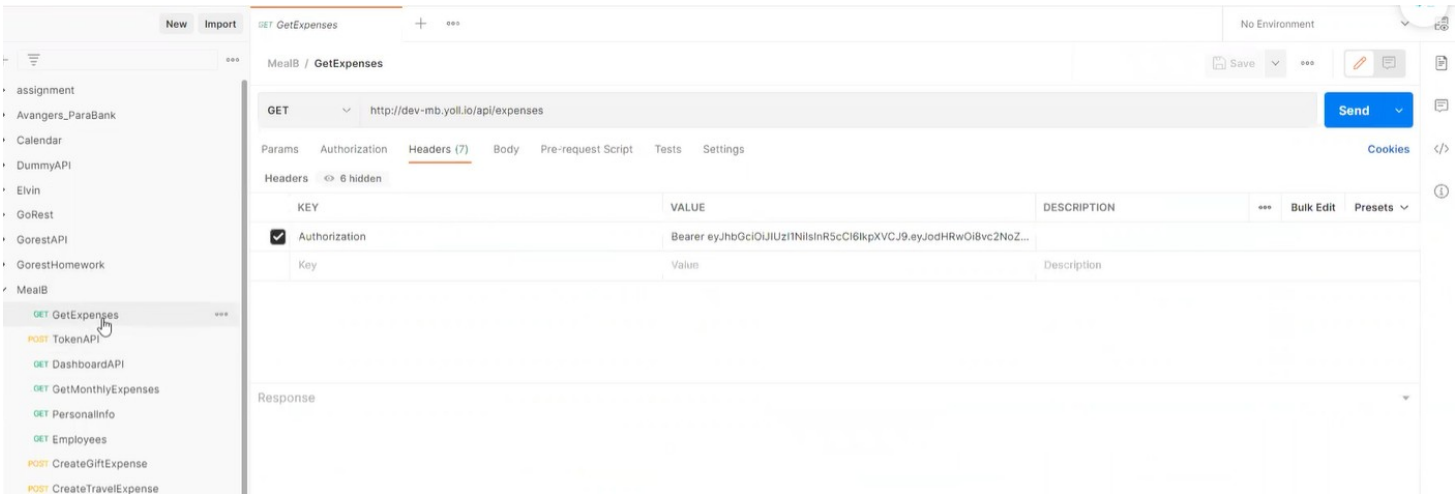
Add a new key-value pair:

- **Key**: Authorization
- **Value**: Bearer YOUR_TOKEN

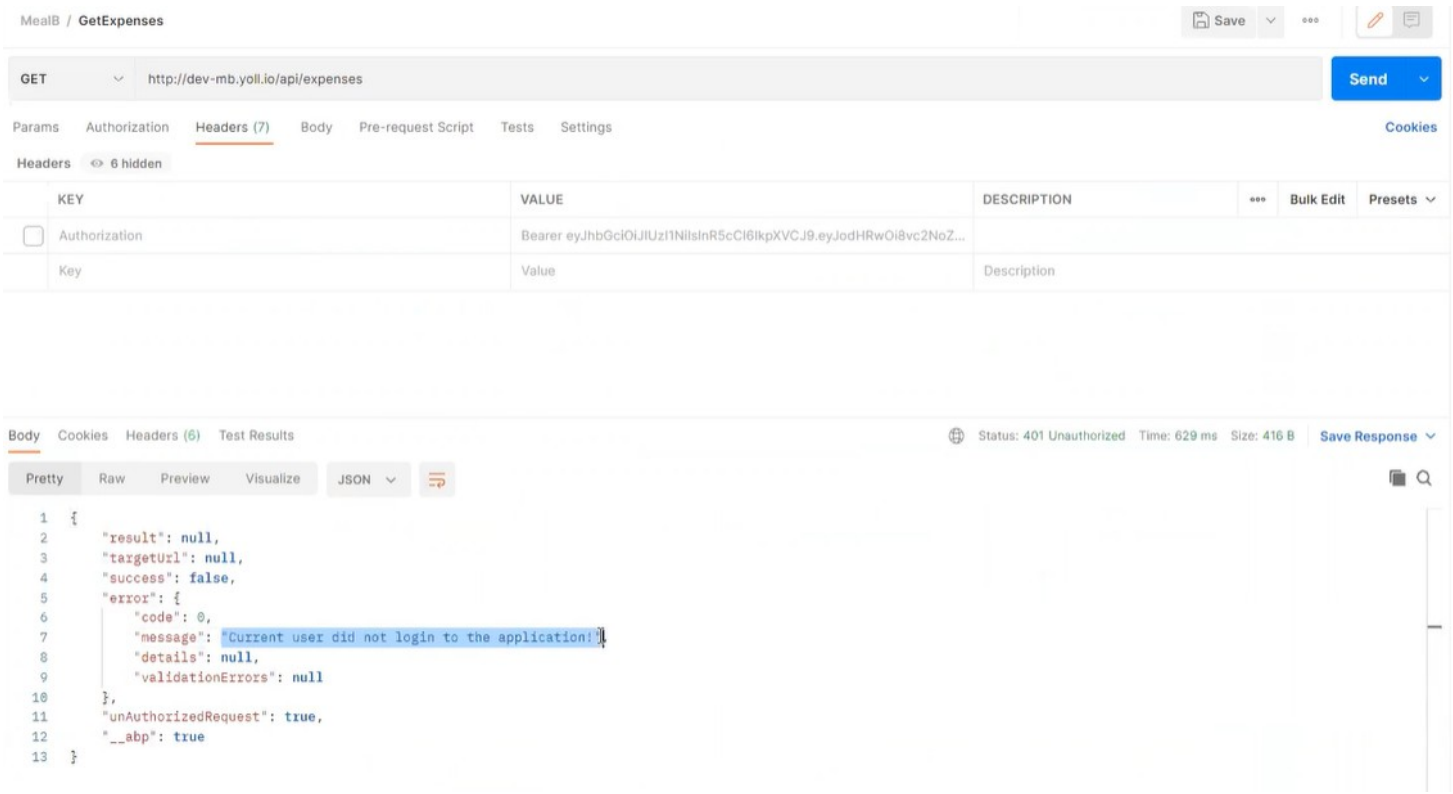


In Postman:

- Go to **Authorization** tab.
- Paste your token: eyJhbGciOiJIUzI1NiIsIn...



If you do not use the Bearer token, the result will be invalidated



Manual Token Retrieval from Login (If API Requires Login)

How to get the Token Key from Postman ?

1. Create a **POST** request to the **login** endpoint.

Example body:

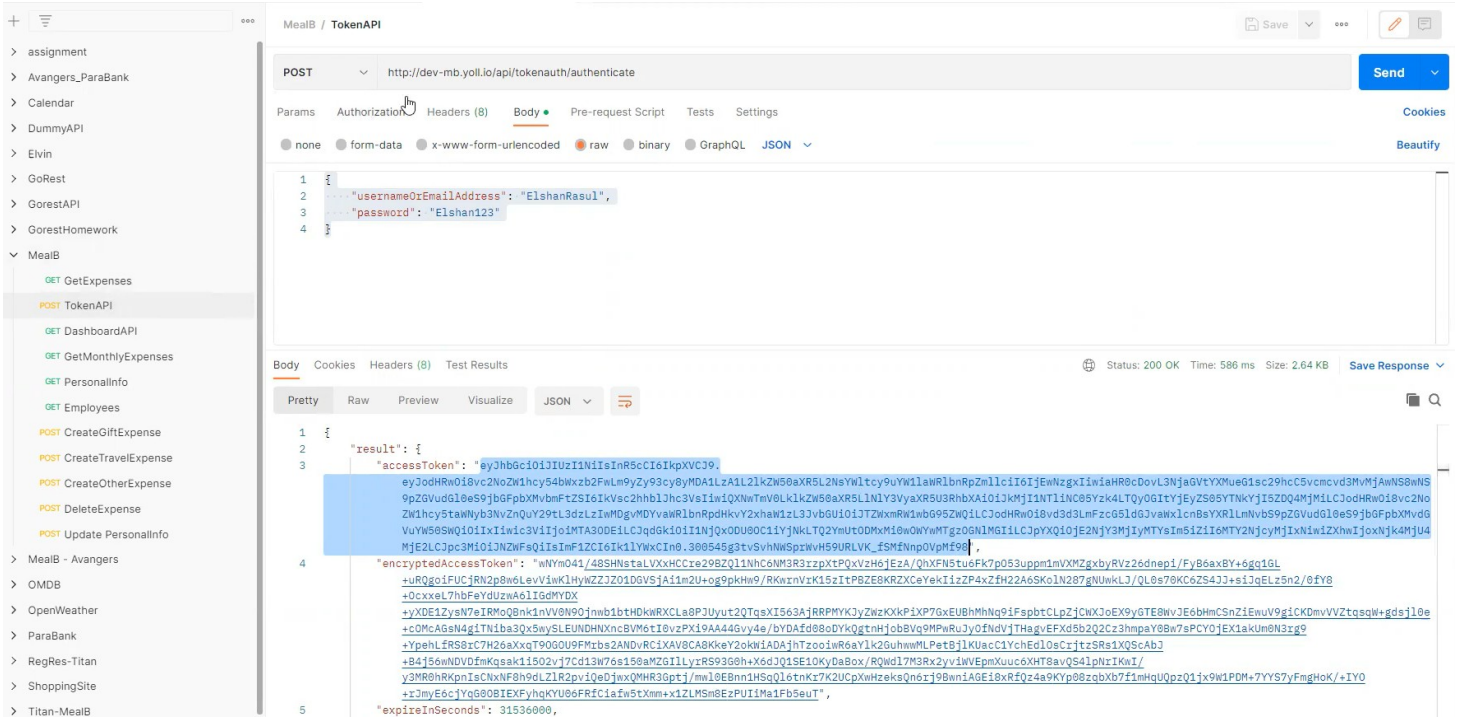
Go to Body (RAW) to check that you have username and password

```
{
  "username": "your_username",
  "password": "your_password"
}
```

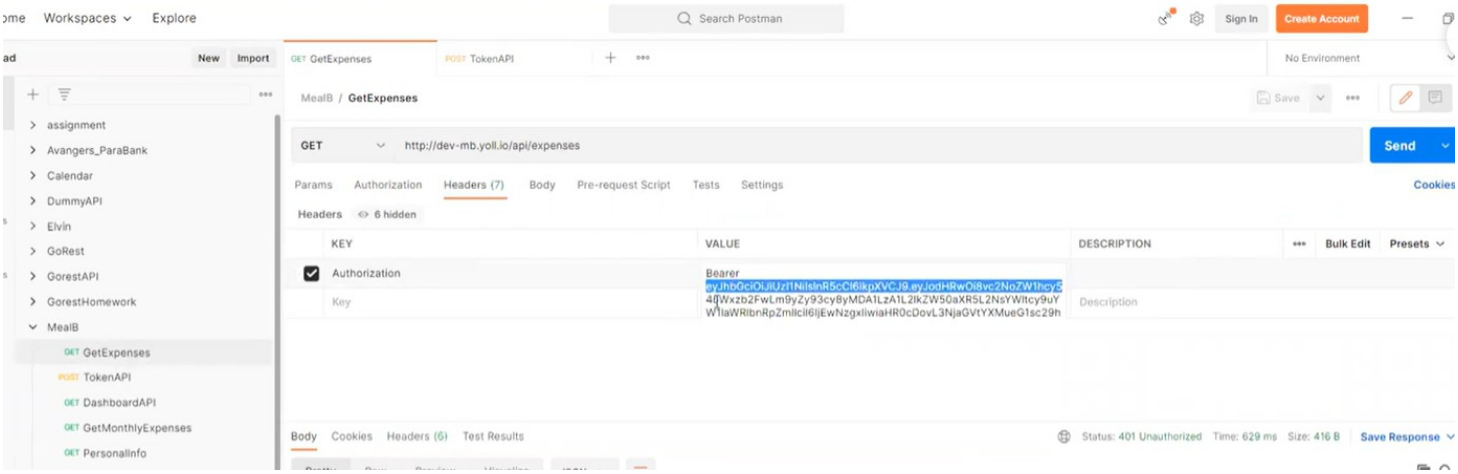
2. Hit **Send**—in the response, you'll see a field like:

```
{
  "token": "your_generated_token"
}
```

3. Copy the token and use it in your **Authorization** header:



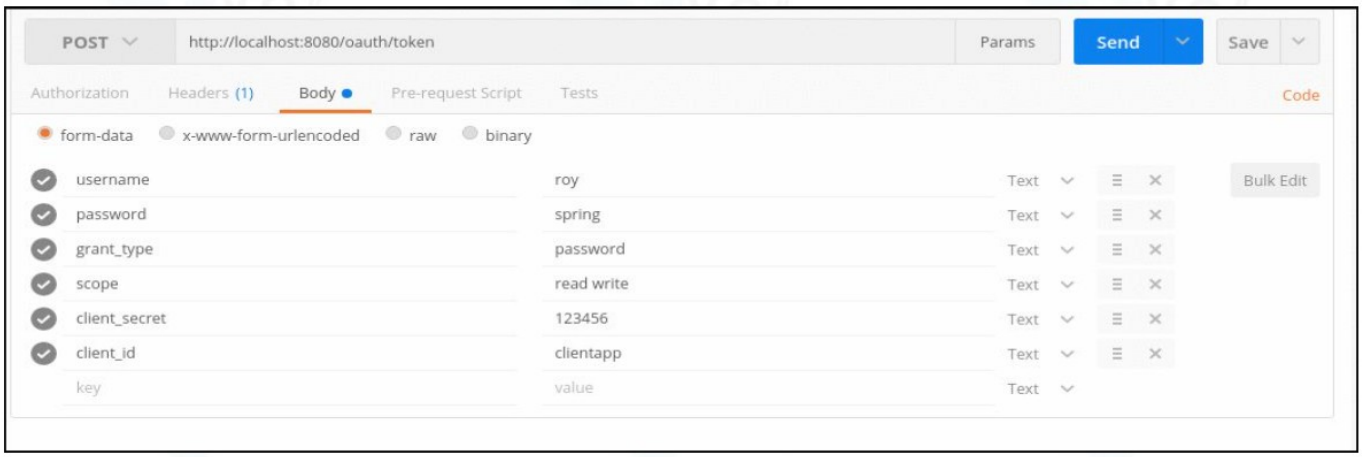
Then paste the bearer Token, and put before the token the word: Bearer



In some companies, we need to use not only the username and password, but also : grant_type, scope, client_secret…

OAUTH TOKEN

For Auth token there is a dedicated URL. User should send required data like (**username**, **password**, **grant_type**, **client_secret**, **client_id**) to that URL.



3. API Key

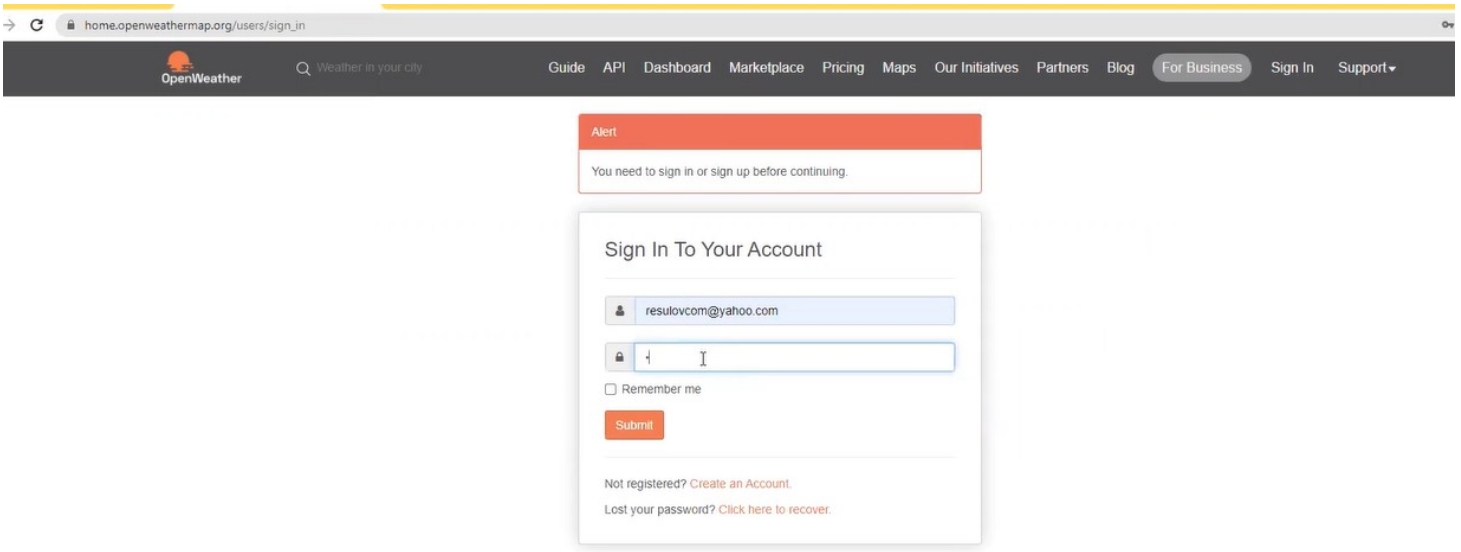
- A unique key identifies and authenticates the user.

Example: Accessing ride fares.

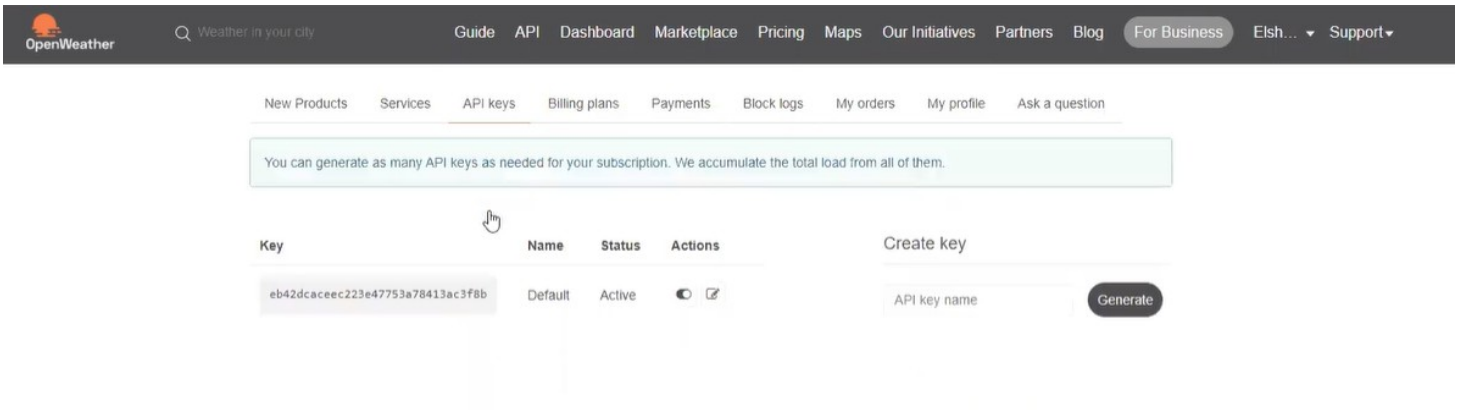
In Postman:

- Go to **Authorization** tab.
- Select **API Key**.
- Key: api_key
- Value: abc123xyz
- Choose where to add it (Header/Query Params).

Get your unique API Key from this Website (Fake)


home.openweathermap.org/users/sign_in
OpenWeather
Weather in your city
Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business Sign In Support
Alert
You need to sign in or sign up before continuing.
Sign In To Your Account
resulovcom@yahoo.com
Remember me
Submit
Not registered? Create an Account.
Lost your password? Click here to recover.

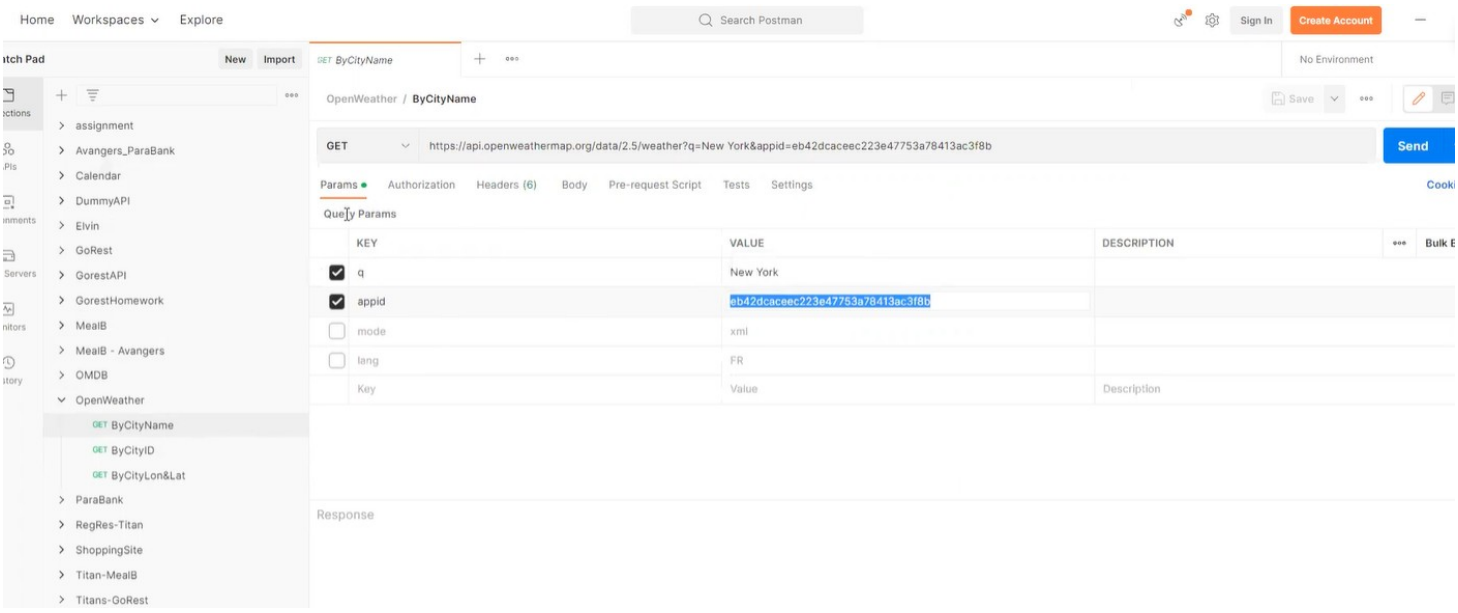
Go to API Keys, Copy the Key


OpenWeather
Weather in your city
Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business Elsh... Support
New Products Services API keys Billing plans Payments Block logs My orders My profile Ask a question
You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.
Key Name Status Actions
eb42dcaceec223e47753a78413ac3f8b Default Active
Create key
API key name
Generate

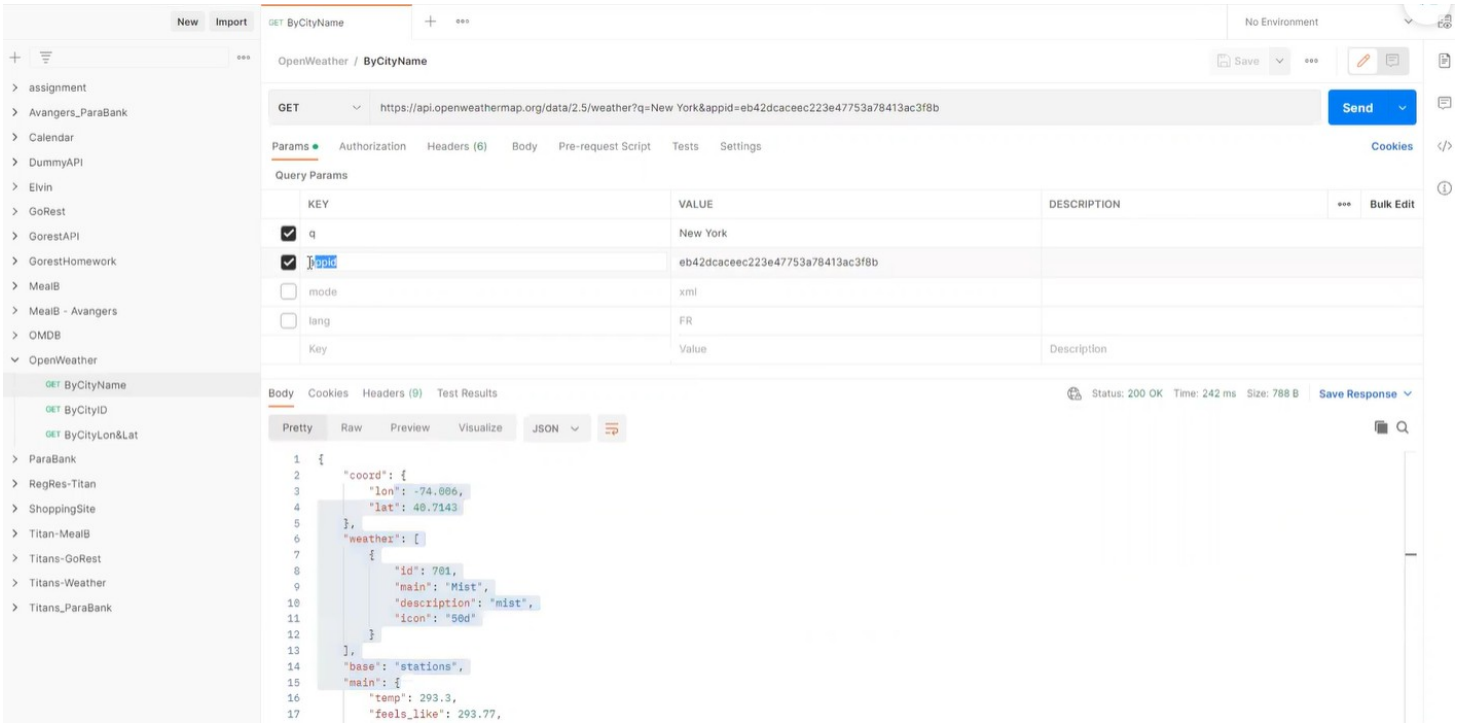
Go to Params, Paste the key

It can also be under Heasders Tag

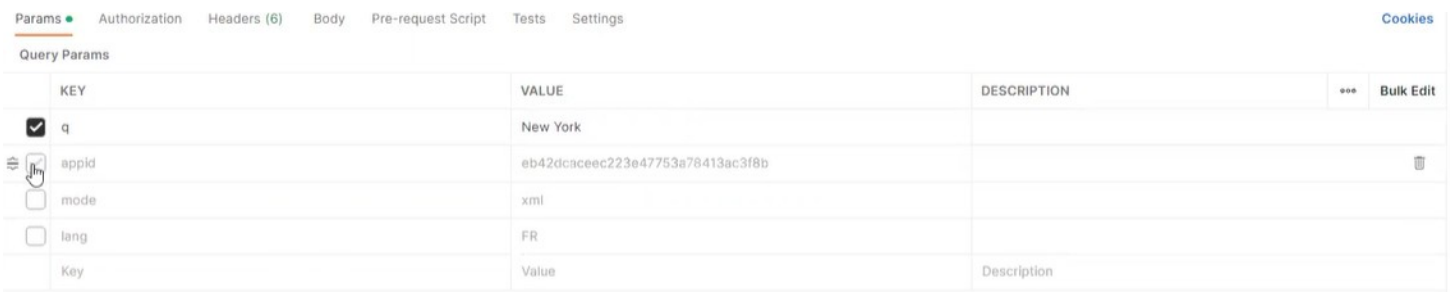
Appid :.....

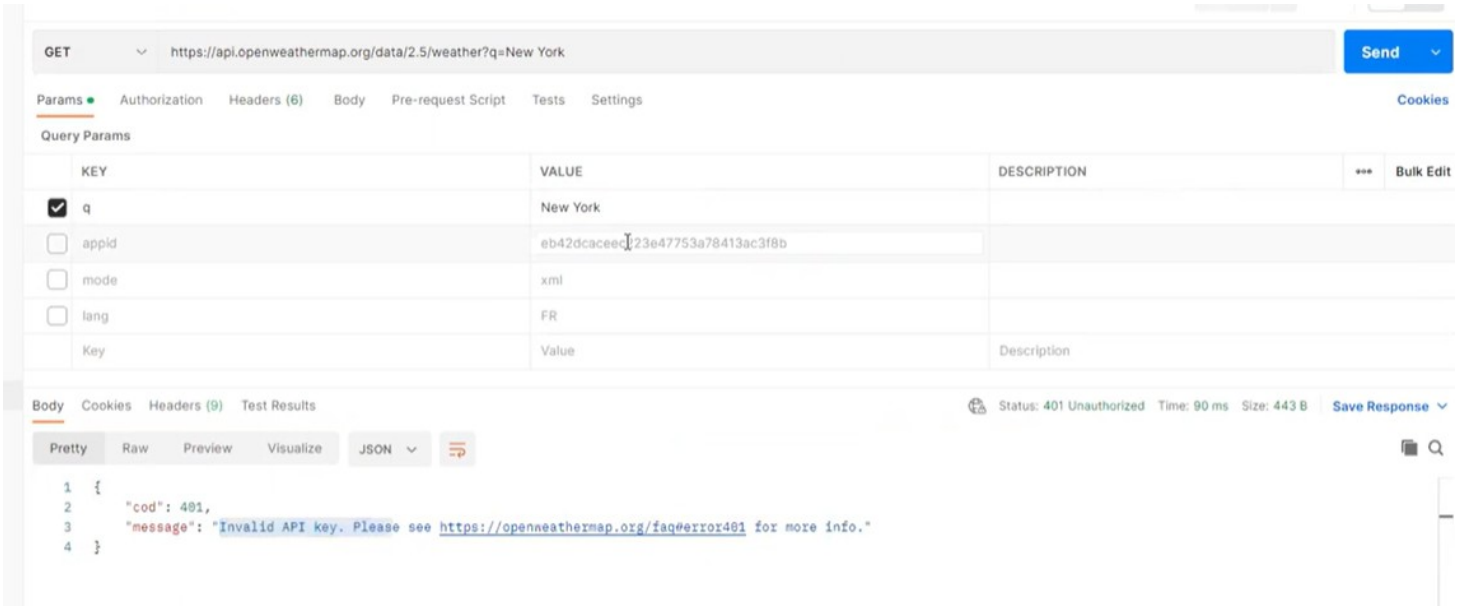


Send the request, and the result:



You can deactivate it, but the result will be invalidated:





Security Measures of Rest API

- Unlike Web applications, RESTful APIs are usually stateless
  - Statelessness means that servers do not save client data between requests.
- Therefore you have to enable some sort of credentials to authenticate. Common practice is to send a secret Access token with each request to authenticate a user

Security Measures of REST API Explained:

1. **Statelessness in REST APIs:**
   - RESTful APIs follow the **stateless** principle, meaning the server does not store any information about the client's previous requests.
   - Each request from a client to the server must contain all the information needed to understand and process the request.
   - **Example:** If a user logs in, their session is not stored on the server. Each subsequent request must include the authentication token to verify their identity.
2. **Authentication Using Credentials:**

- Since REST APIs are stateless, each request must be authenticated independently.
- **Access tokens** (such as JSON Web Tokens - JWT) are commonly used to identify and authenticate users without maintaining session data on the server.

3. **How Authentication Works:**
   - When a user logs in, the API issues an **access token**.
   - For every request, the client sends this token in the **Authorization header** to verify the user's identity.

Example Request:

sql

CopyEdit

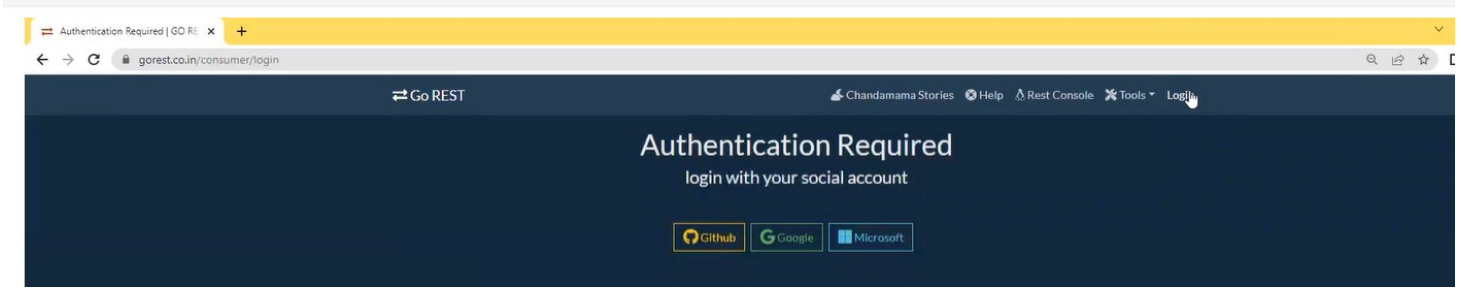
GET /user/profile HTTP/1.1

Host: api.example.com

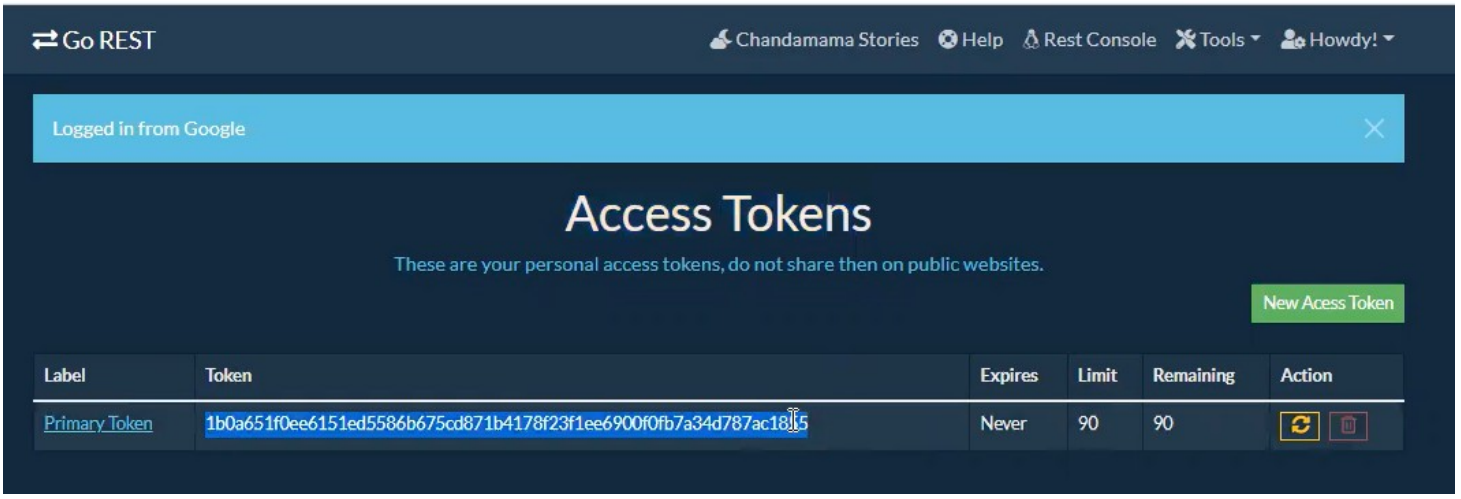
Authorization: Bearer your_access_token

4. **Why Statelessness is Secure:**
   - Reduces the risk of session hijacking since no session data is stored on the server.
   - Ensures better **scalability** because each request is independent and does not rely on prior information.

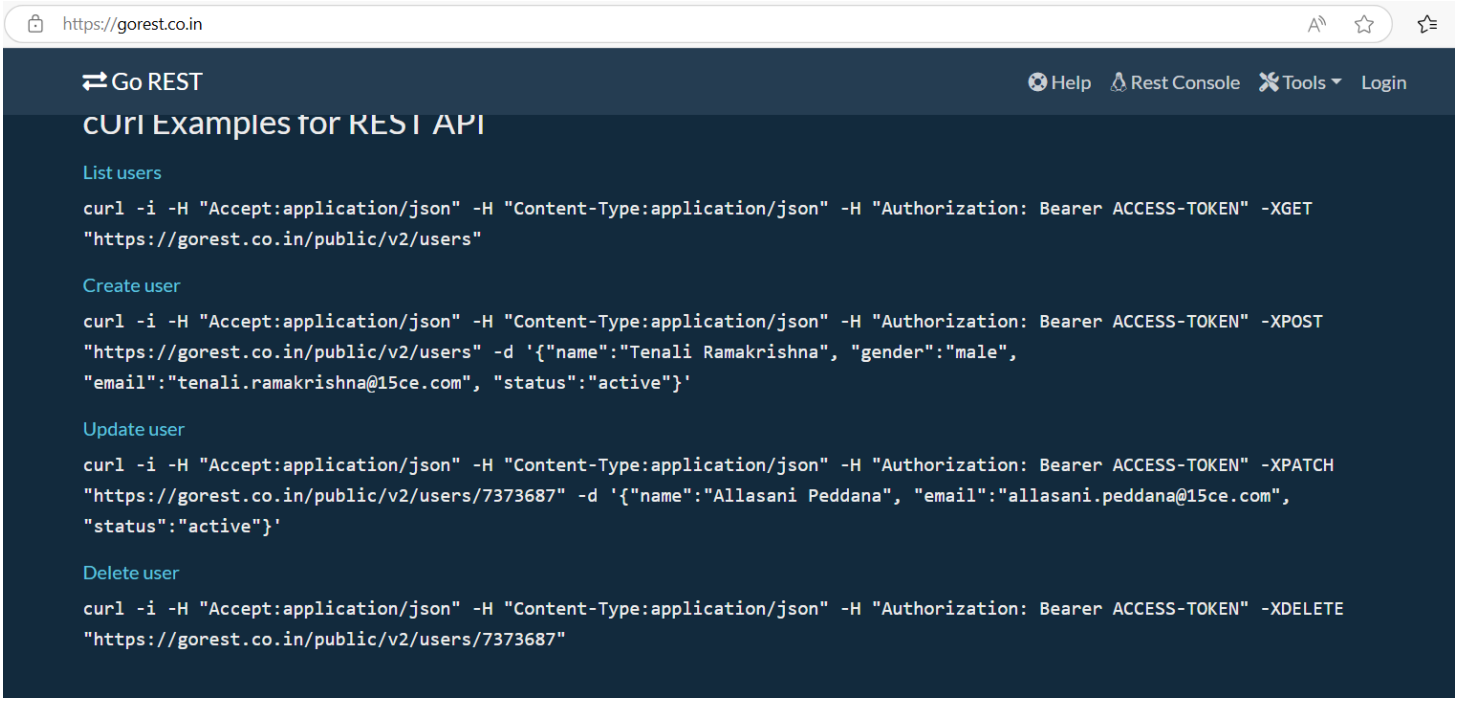
Use this link, and create an account, then choose Google:



Then the token is created:

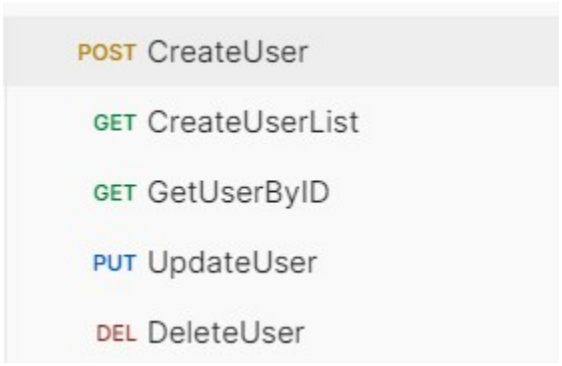


[GraphQL and REST API for Testing and Prototyping | GO REST]


https://gorest.co.in
Go REST
Help
Rest Console
Tools
Login
cURl Examples for REST API
List users
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer ACCESS-TOKEN" -XGET "https://gorest.co.in/public/v2/users"
Create user
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer ACCESS-TOKEN" -XPOST "https://gorest.co.in/public/v2/users" -d '{"name":"Tenali Ramakrishna", "gender":"male", "email":"tenali.ramakrishna@15ce.com", "status":"active"}'
Update user
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer ACCESS-TOKEN" -XPATCH "https://gorest.co.in/public/v2/users/7373687" -d '{"name":"Allasani Peddana", "email":"allasani.peddana@15ce.com", "status":"active"}'
Delete user
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer ACCESS-TOKEN" -XDELETE "https://gorest.co.in/public/v2/users/7373687"

Post Method

Must be on the first rank



Use the same Bearer Token for all the methods. Otherwise, the result will be failed !

Create user

curl -i -H "Accept:application/json"

-H "**Content-Type:**application/json" -H

"Authorization: Bearer ACCESS-TOKEN"

-XPOST "https://gorest.co.in/public/v2/users" -d

'{"**name**":"Tenali Ramakrishna",

"**gender**":"male",

"**email**":"tenali.ramakrishna@15ce.com",

"**status**":"active"}'

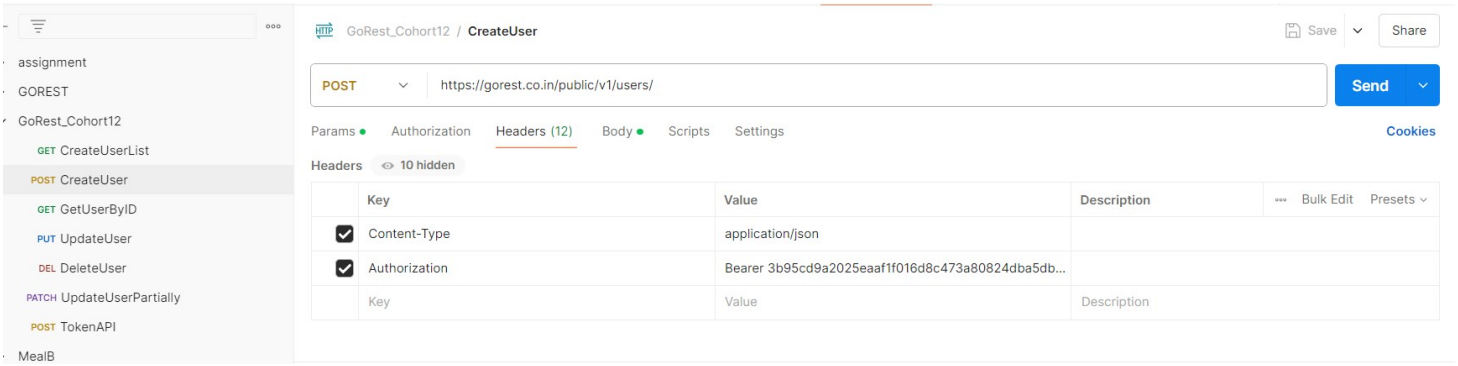
Use this url:

https://gorest.co.in/public/v2/users

On Headers Tag:

We need to put the same Content Type on Headers from : application/json

And put the Bearer Token on value, and choose Authorization on Key



On Body / Raw Tag:

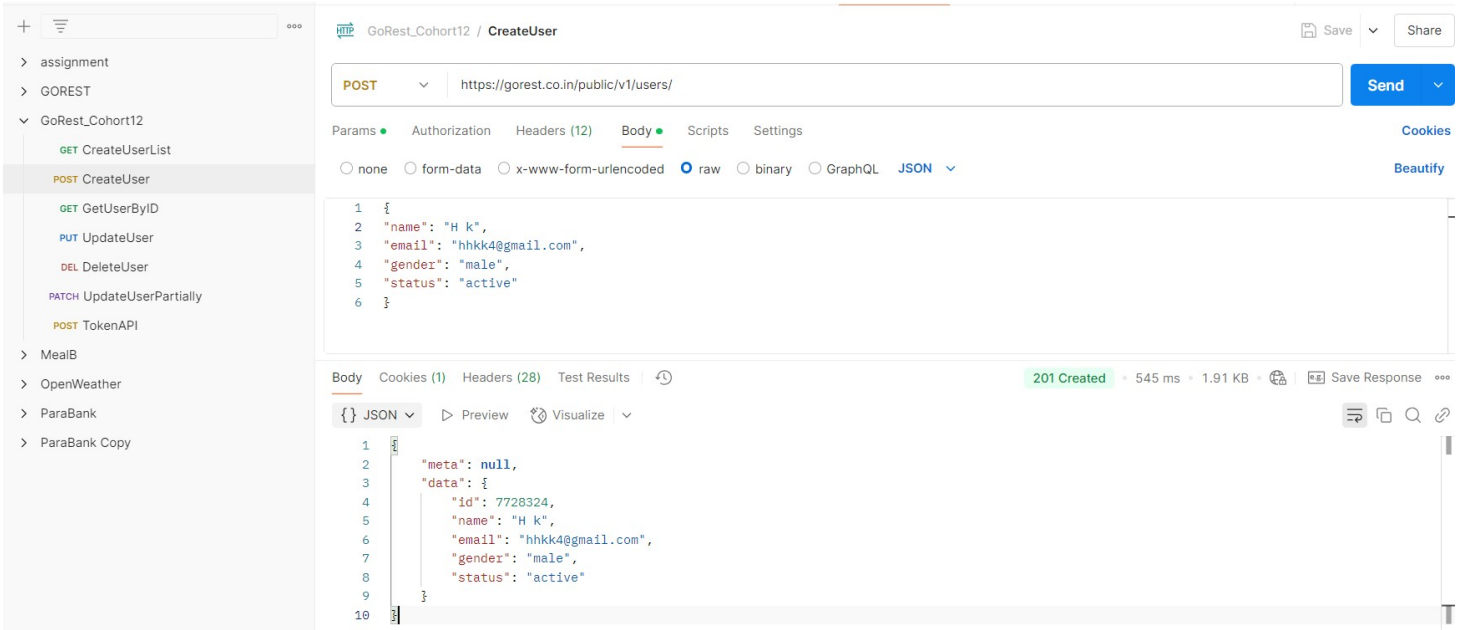
Put the same order and variables for the data structure

'{"name":".....",

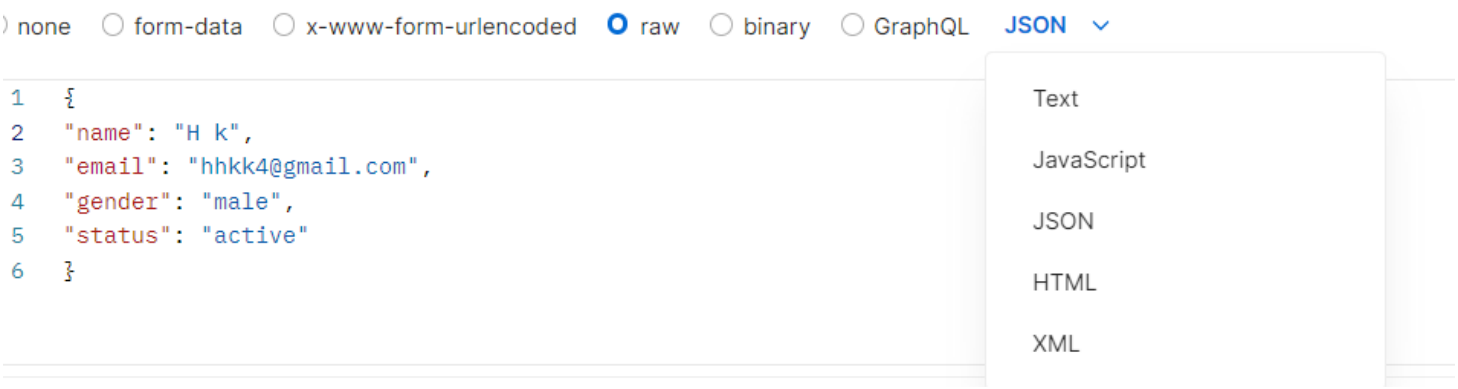
"gender":".....",

"email":".....",

"status":"....."}



And select Json:

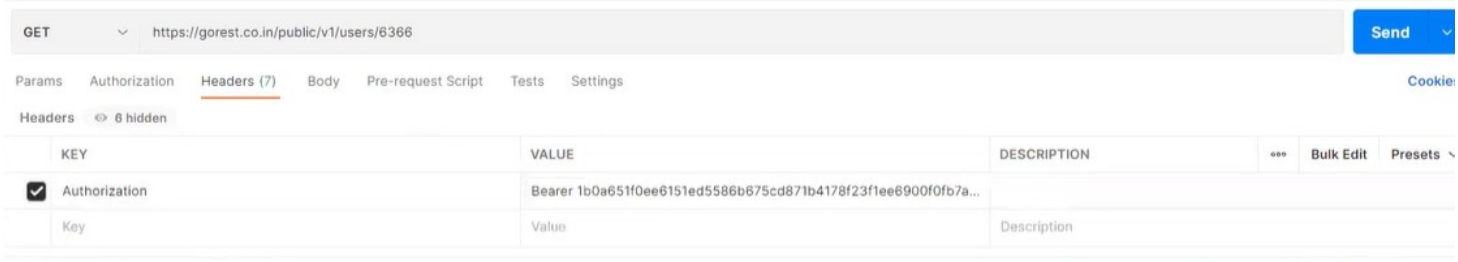


You do not need to put the id on the data structure because the system will generate it automatically !

Use only one email address for each user, and do not duplicate it with other users. It will not be accepted.

Get Method

Pass the same Bearer Token:

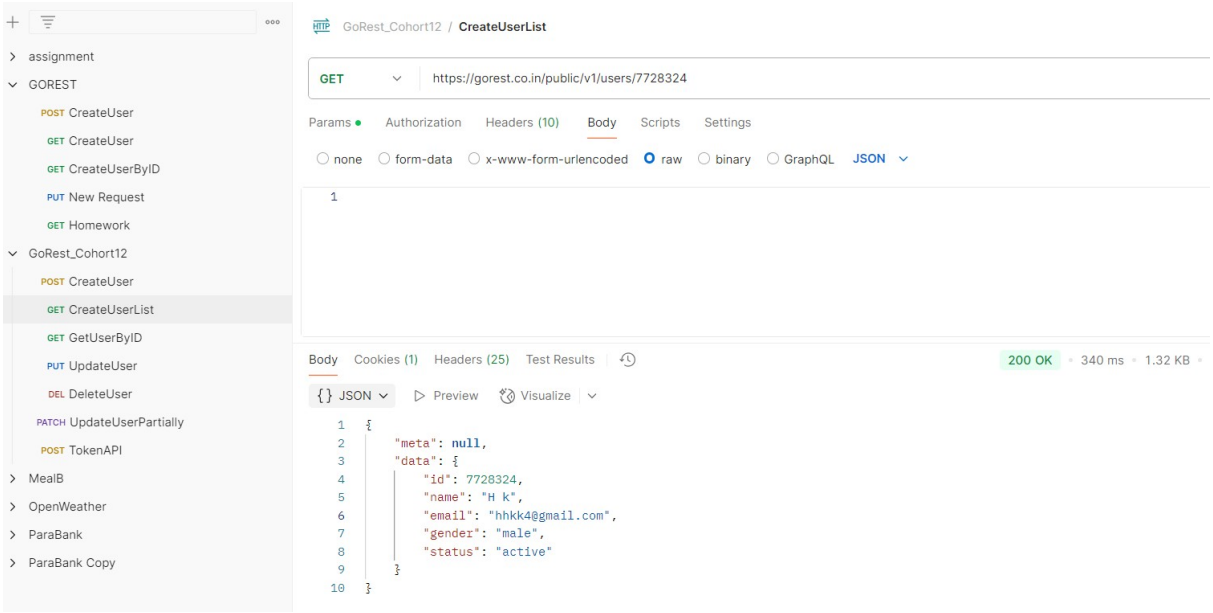


If not, the result will be Invalid

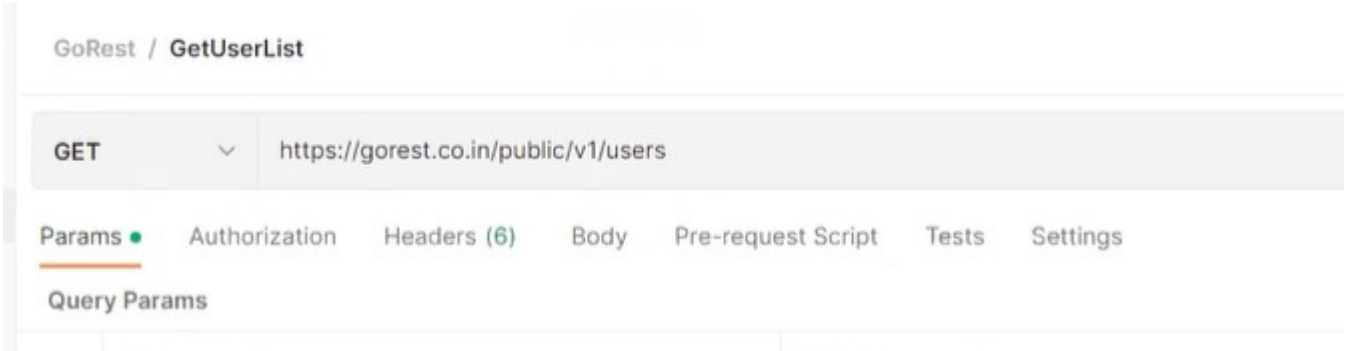
Use the url and the parameter (7728324) after the slash (/)to get this user:

https://gorest.co.in/public/v1/users/7728324

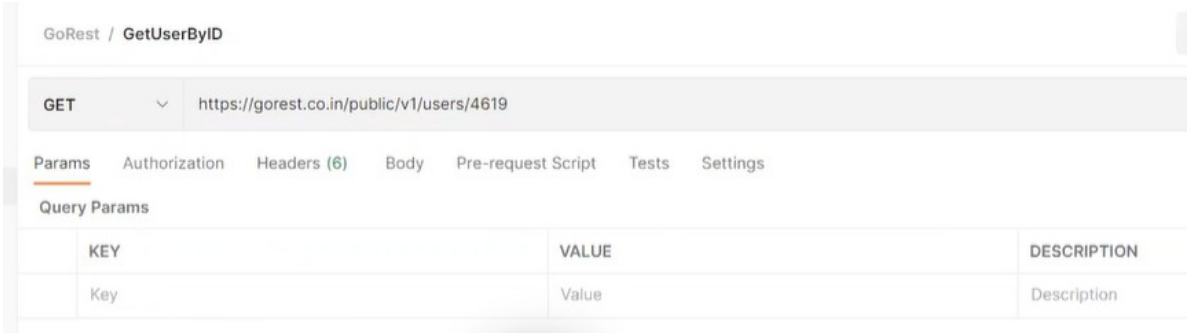
Send the request



Get All the users:



Get the user: 4619



Put Method

Specify the user id:

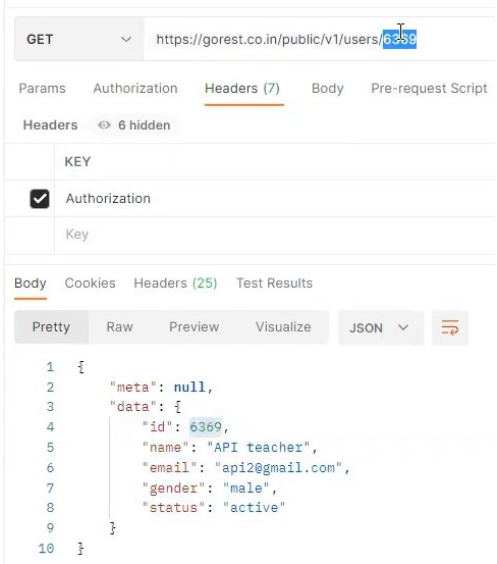
Use the url and the parameter (6369) after the slash (/)to get this user:

https://gorest.co.in/public/v1/users/6369

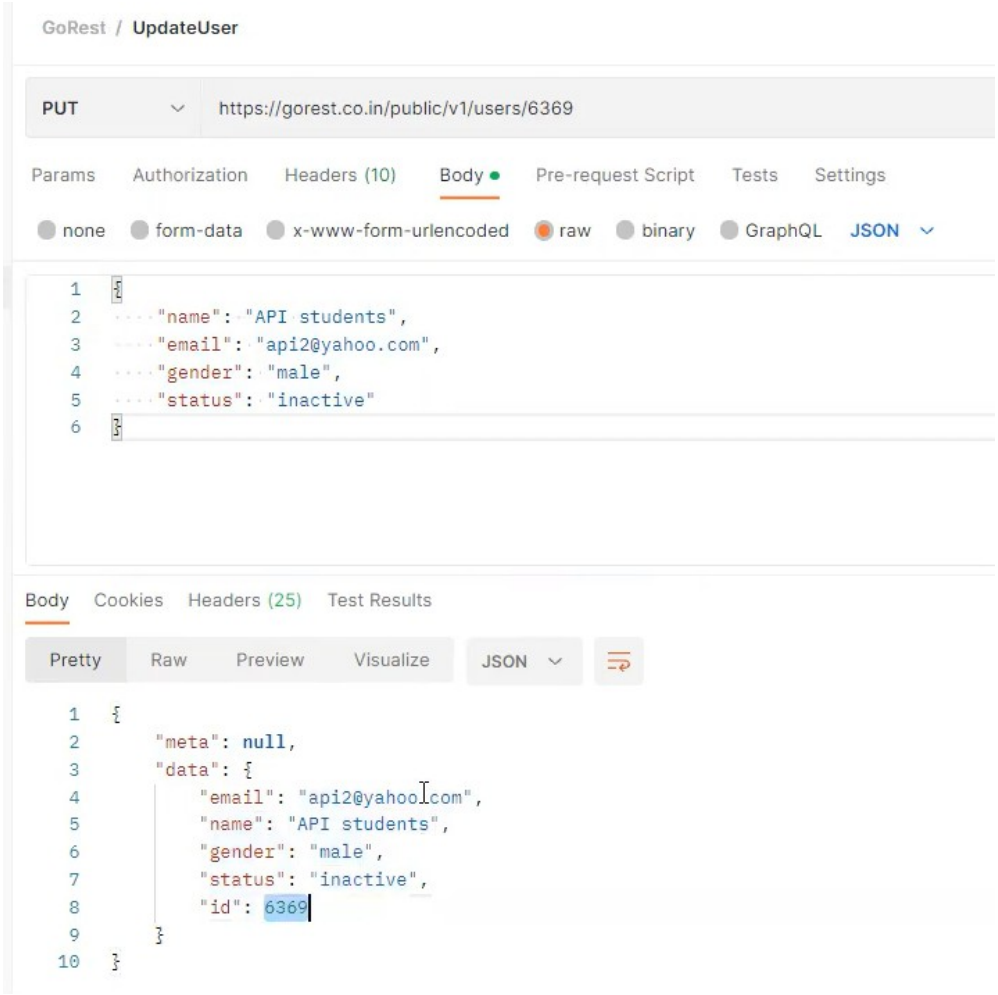
Send the request

In get method :

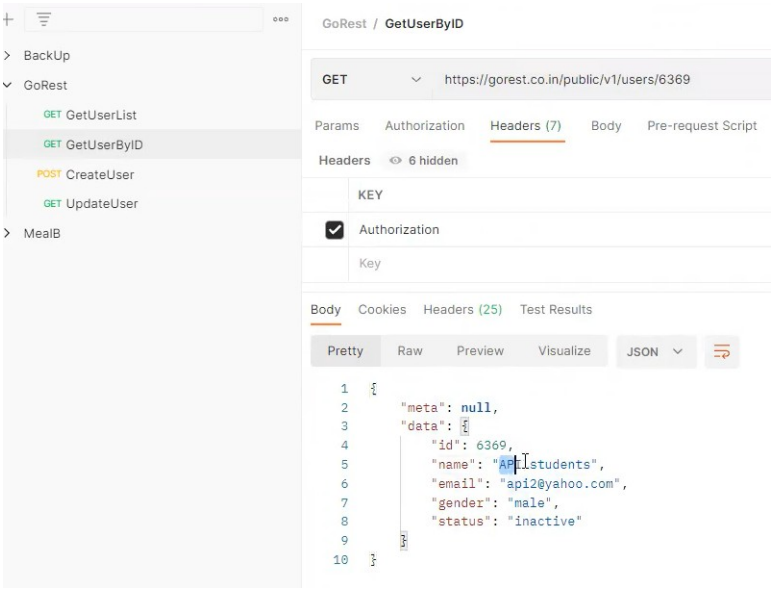
We need to update the name and the status of the user: 6369



In put method:



To see if the user 6369 data are updated or not, USE GET METHOD



PUT and PATCH Methods

The PUT and PATCH methods are both used to update resources in REST APIs, but they work in slightly different ways.

PUT Method:

- **Purpose: Replaces the entire resource with a new one.**
- **Behavior: If a resource already exists, it updates the whole resource.**
- **Requires all fields, even if you're only updating one.**

- **Example**: **Updating a user's information**:

```
PUT /users/123

{
  "name": "John Doe",
  "email": "john@example.com",
  "gender": "male",
  "status": "active"
}
```

PATCH Method:

- **Purpose: Partially updates a specific field(s) of a resource.**
- **Behavior: Only the fields provided will be updated. Other fields remain unchanged.**
- **Only requires the fields you want to update.**

- **Example**: **Updating only the user's** status:

```
PATCH /users/123
```

Request Body:

```
{
  "status": "inactive"
}
```

This updates **only** the status and leaves other user details unchanged.

Specify the user id:

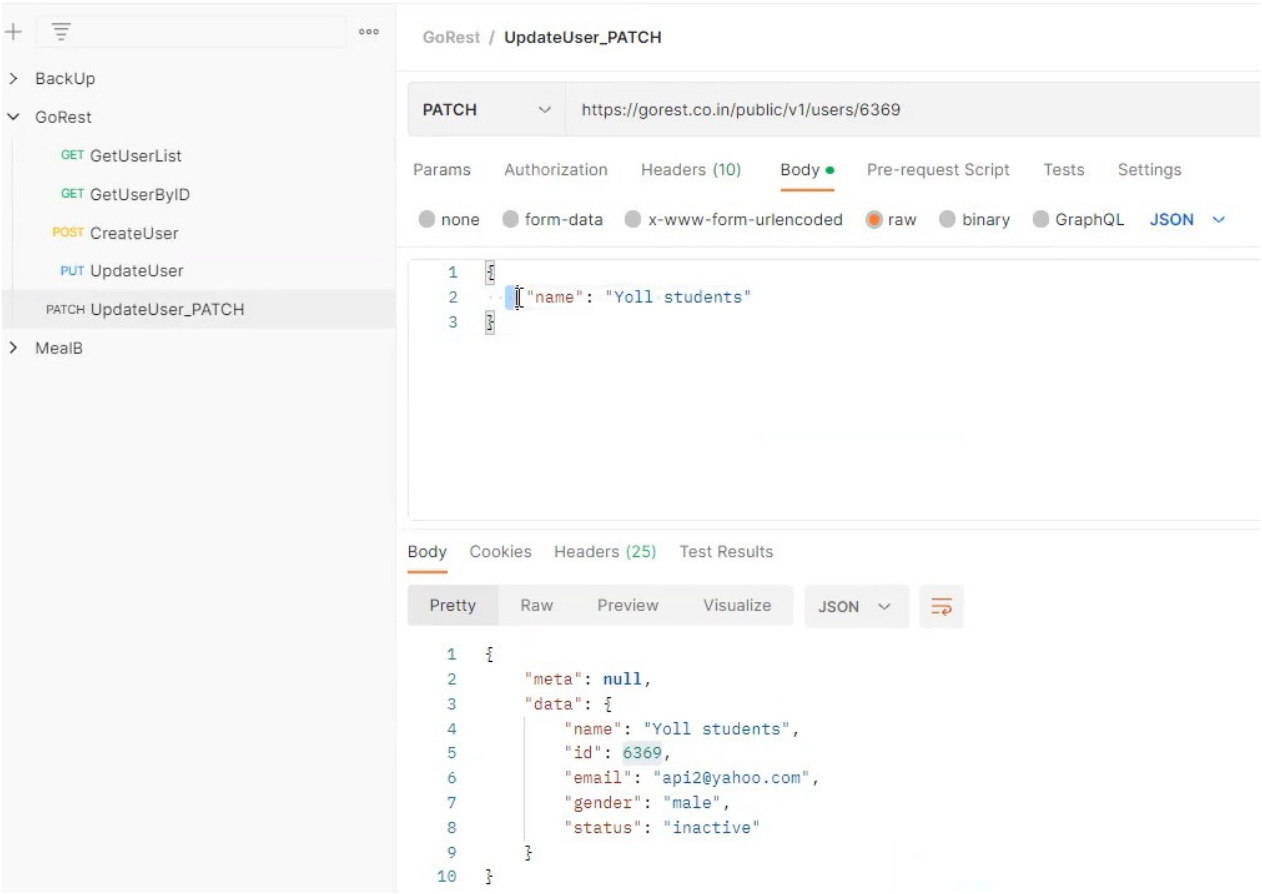
Use the url and the parameter (6369) after the slash (/)to get this user:

https://gorest.co.in/public/v1/users/6369

Send the request

If we need to change the name of the user 6369, we can only change the name not the whole information.

If you need to change more than one element like Name and status, you can only update these two elements.



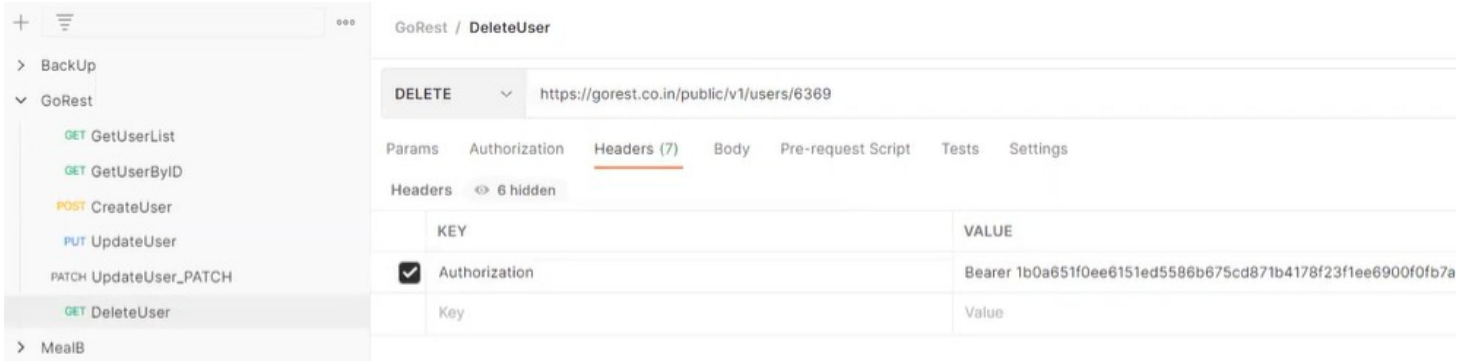
Delete Method

Specify the user id:

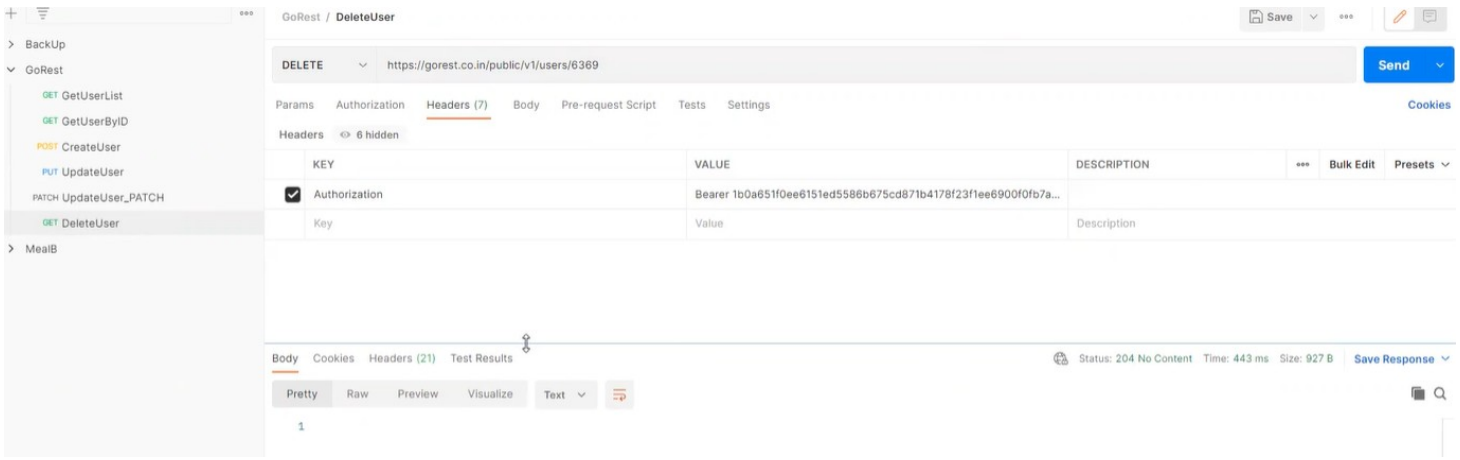
Use the url and the parameter (6369) after the slash (/)to get this user:

https://gorest.co.in/public/v1/users/6369

Send the request

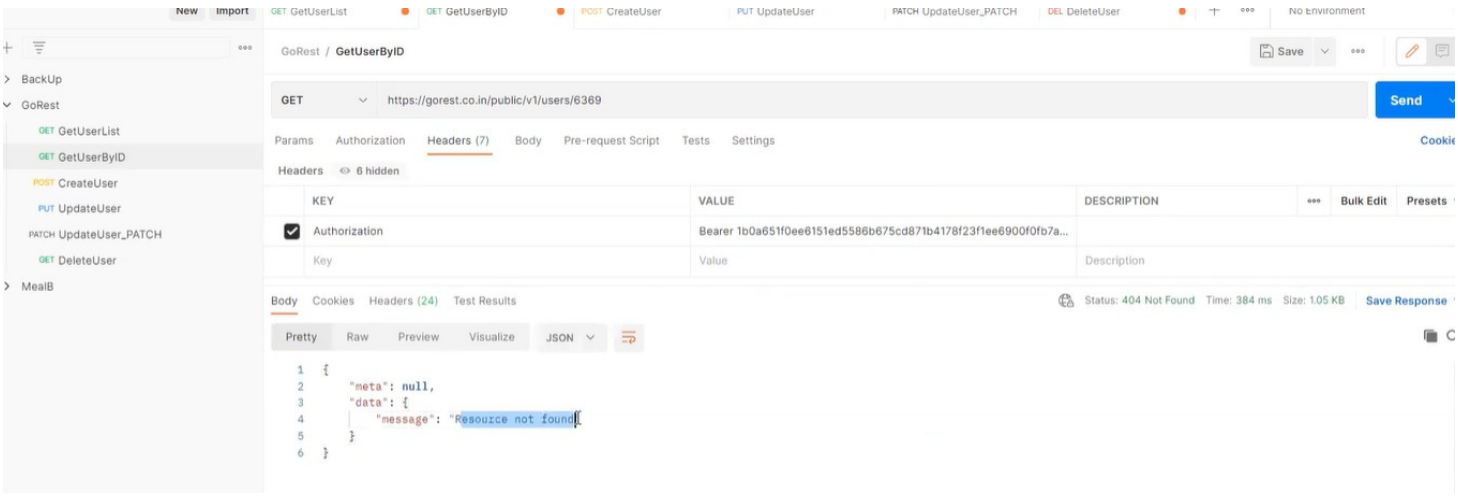


When you send the request, there is no result:



To see if the user 6369 data are deleted or not, USE GET METHOD:

Ressources not found which means 6369 is deleted.

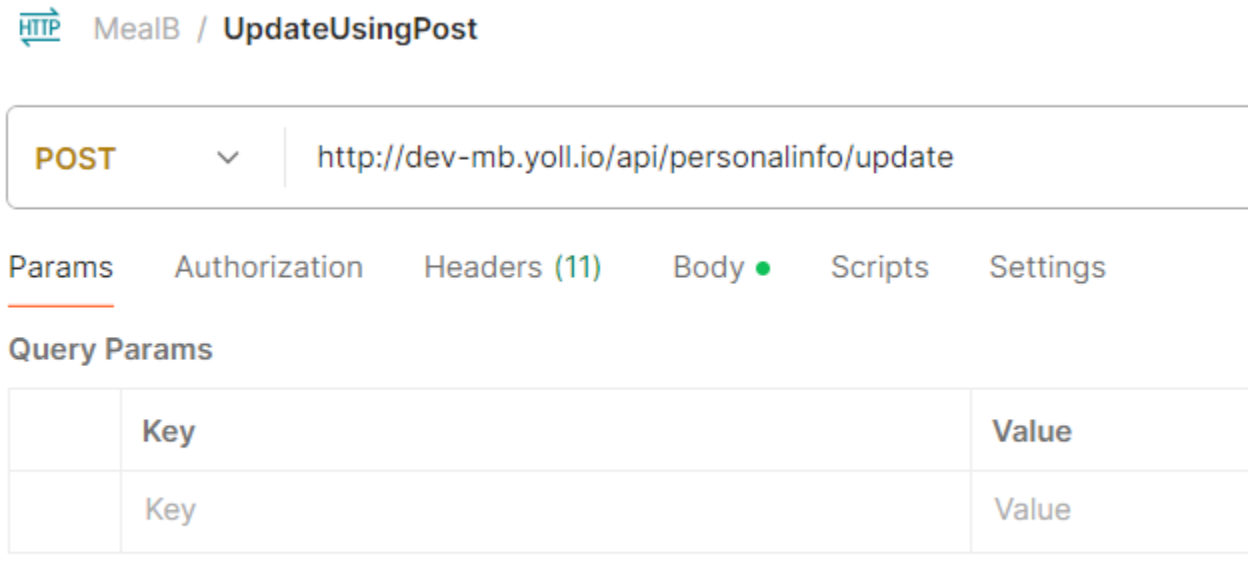


Always delete one by one, not the users because it is very risky !

WE can DELETE AND PUT BY USING ONLY POST METHOD

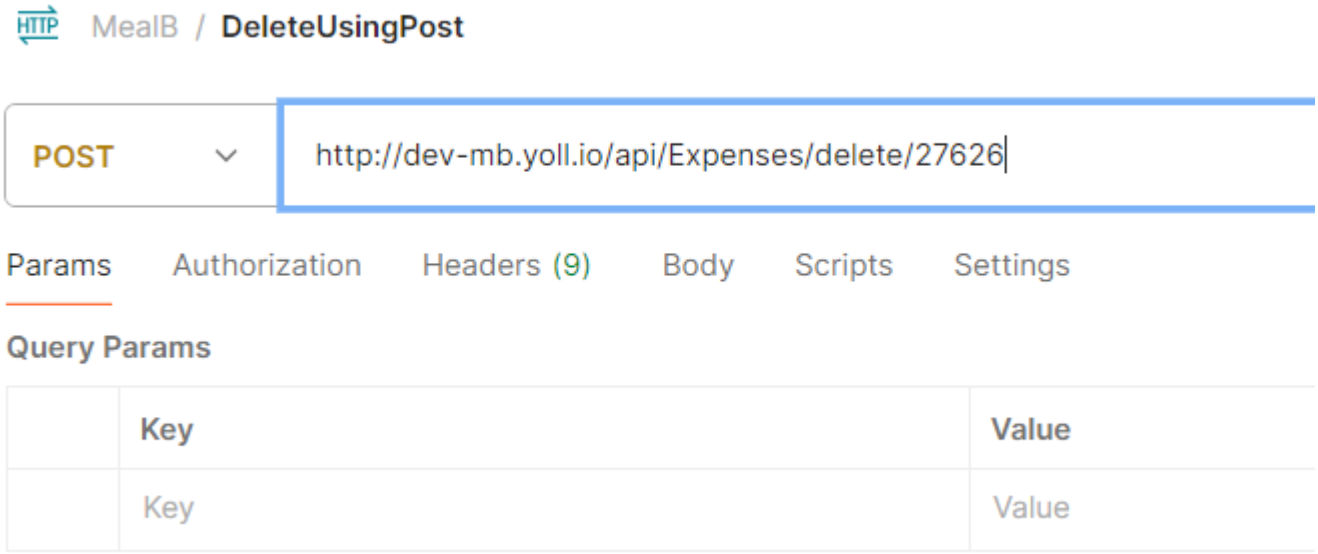
1- UPDATE:

http://dev-mb.yoll.io/api/personalinfo/update



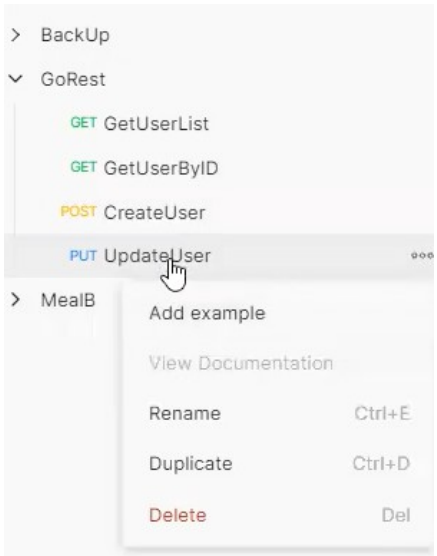
2- DELETE:

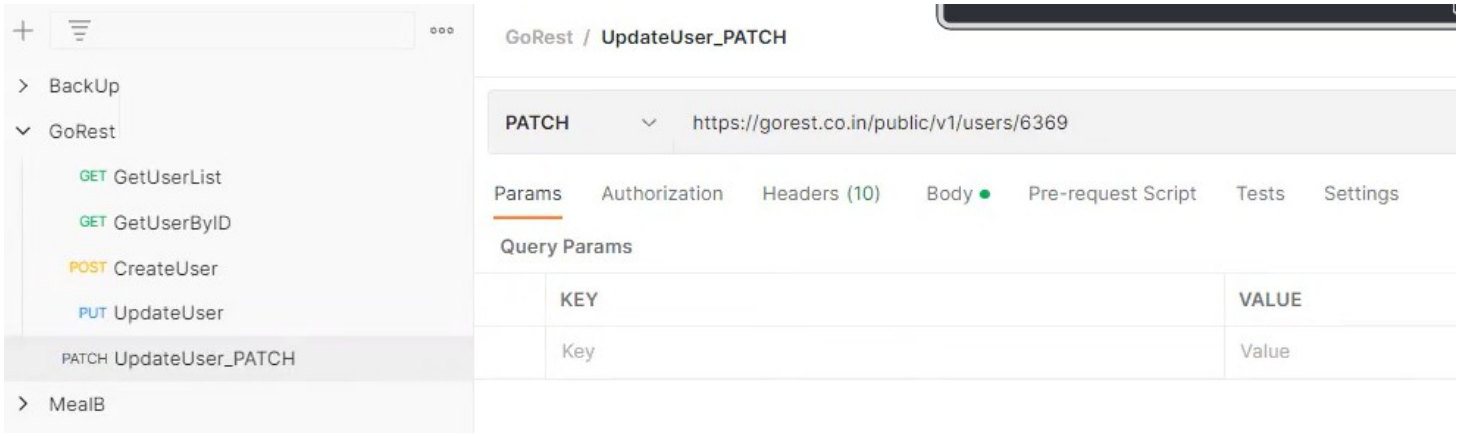
http://dev-mb.yoll.io/api/Expenses/delete/27626



Duplicate a method

Duplicate a method and keep the original one unchanged, then edit the name and the method. Save it:





Metadata information:

The provided JSON snippet represents metadata information, particularly **pagination** details from an API response. This structure is commonly used in RESTful APIs to manage large sets of data by breaking them into smaller, more manageable chunks (pages).


```
meta": {
        "pagination": {
            "total": 2946,
            "pages": 295,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"

            }
        }
```


Field Breakdown:

1. **`meta`**: This is the parent object, typically used to hold metadata about the API response.
2. **`pagination`**: Contains information about the paginated data.

🔹 Pagination Details:

Field	Description	Example Value
`total`	Total number of records available in the dataset.	`2946` – Total users in the system.
`pages`	Total number of pages available based on the limit.	`295` – Total pages when paginating.
`page`	The current page being viewed or requested.	`1` – You are on page 1.

Field	Description	Example Value
`limit`	Number of records displayed per page.	`10` – Each page shows 10 user records.
`links`	URLs to navigate between pages (previous, current, next).	Contains navigation links.
`previous`	Link to the previous page (if available).	`null` – No previous page (on page 1).
`current`	Link to the current page.	URL for the current page (`page=1`).
`next`	Link to the next page (if available).	URL for the next page (`page=2`).

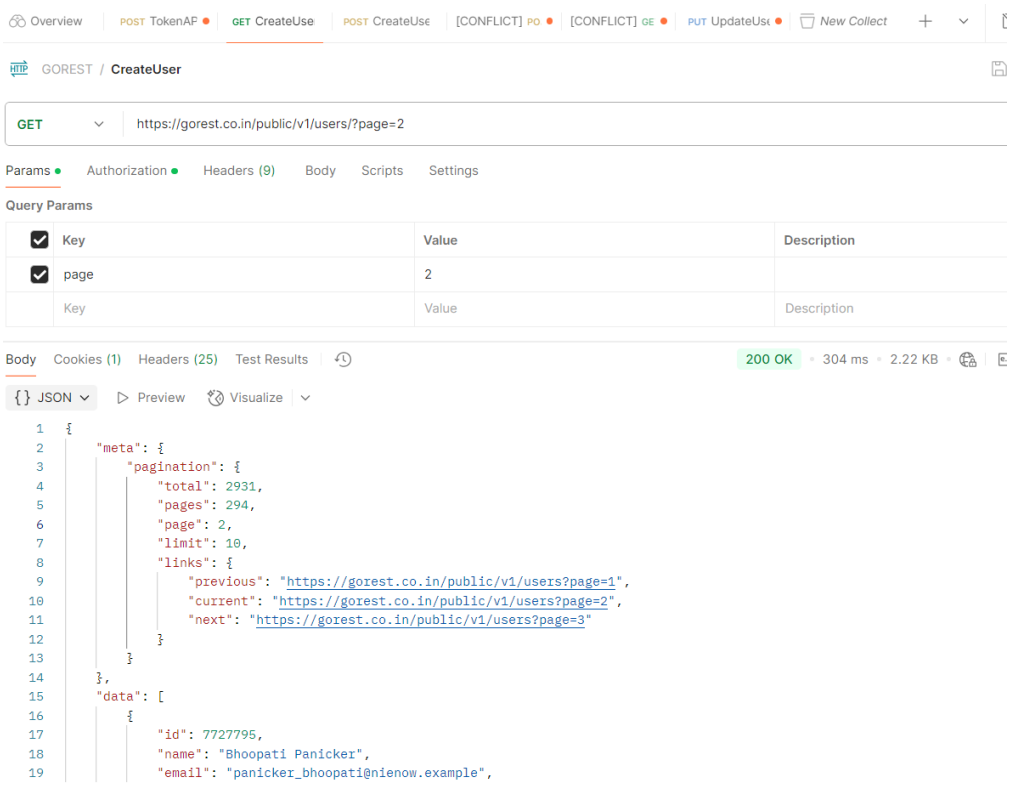
How Pagination Works:

1. **Initial Request (Page 1)**:
   URL: `https://gorest.co.in/public/v1/users?page=1`
   - You receive the first 10 users out of 2946.
   - Since it's the first page, `"previous": null`.
2. **Navigating to the Next Page**:
   Use the `next` link to retrieve page 2:
   URL: `https://gorest.co.in/public/v1/users?page=2`
3. **Last Page**:
   - When on the final page, the `"next"` value will be `null`.
   - Example: If you're on page 295, no further pages exist.

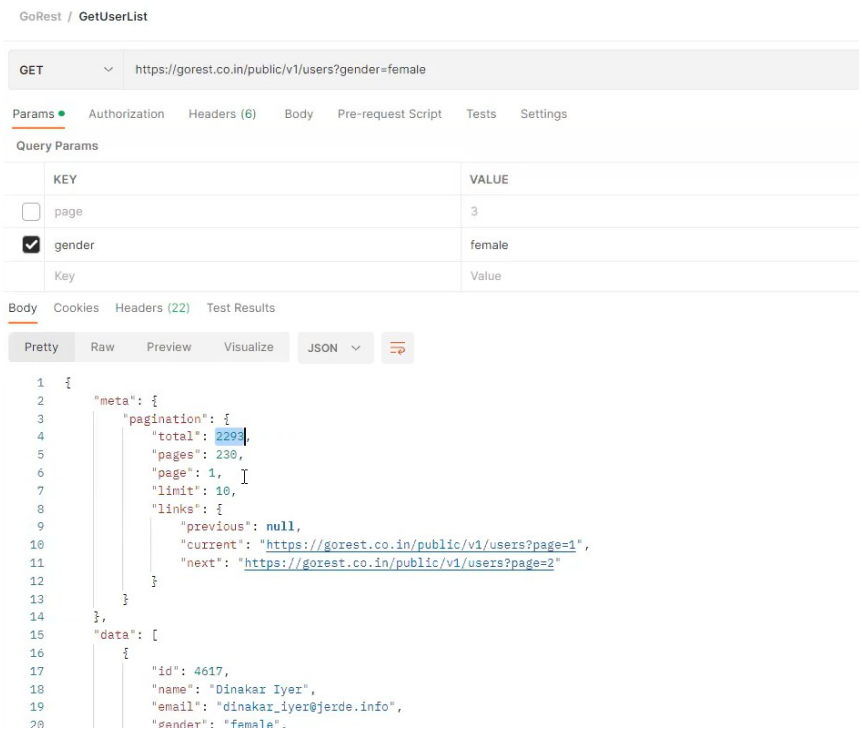
How to filter pages with some criterias ?

To filter pages with specific criteria in a REST API, you typically use **query parameters** in your request URL. Most RESTful APIs support filtering based on various fields like **name**, **status**, **gender**, **date**, etc.

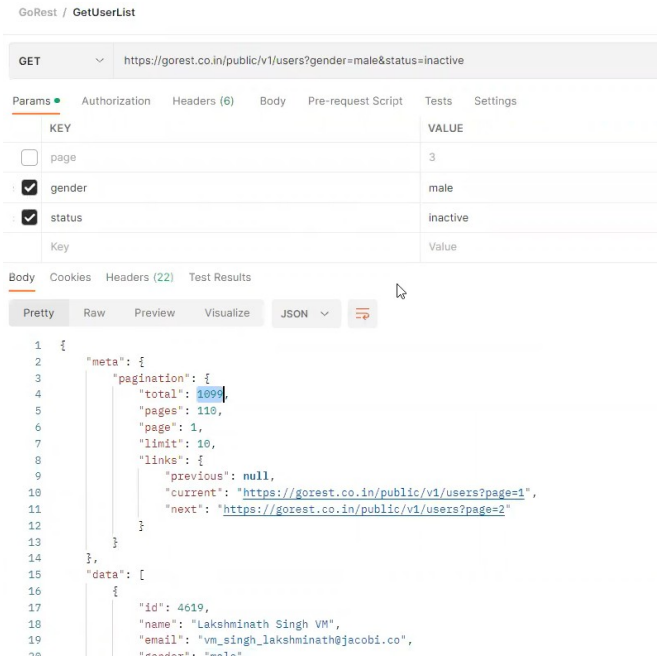
Show the users on Page :2



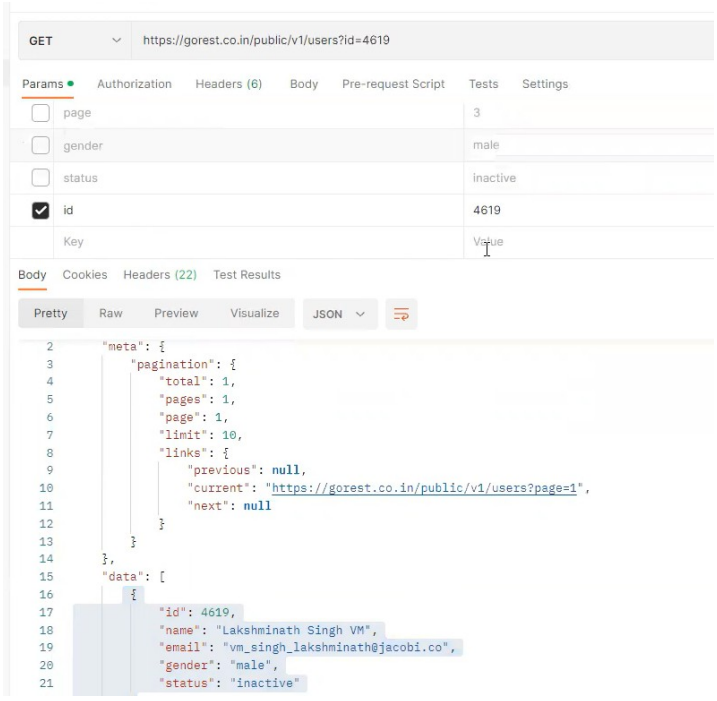
Show only the Female users



Show inactive male:



Get the user with the id: 4619



Common Query Parameters for Filtering:

Filter Type	Example Parameter	Description
Page Number	`?page=2`	Get users from **page 2**.
Page Size (Limit)	`?limit=5`	Return **5** records per page.
Filter by Name	`?name=John`	Return users with the **name "John"**.

Filter Type	Example Parameter	Description
Filter by Status	`?status=active`	Return users with **active** status.
Filter by Gender	`?gender=female`	Return users who are **female**.
Multiple Filters	`?gender=male&status=active`	Return **male** users who are **active**.
Sorting Data	`?sort=name`	Sort users by **name** in ascending order.
Descending Sort	`?sort=-name`	Sort users by **name** in **descending** order.
Date Filtering	`?created_at=2023-01-01`	Filter by users created on **2023-01-01**.

Practical Examples:

1. **Get Active Users on Page 1:**

   ```
   https://gorest.co.in/public/v1/users?page=1&status=active
   ```

2. **Get 5 Female Users on Page 3:**

   ```
   https://gorest.co.in/public/v1/users?page=3&limit=5&gender=female
   ```

3. **Get Users Created After a Specific Date:**

   ```
   https://gorest.co.in/public/v1/users?created_at>=2023-01-01
   ```

4. **Get Male Users, Active Status, Sorted by Name:**

   https://gorest.co.in/public/v1/users?gender=male&status=active&sort=name

Path and Query Parameters in Postman

QUERY PARAMETERS: are passed in KEY + VALUE format.
In automation we use .param(key, value) method to pass query parameters. Query parameters are separated from URL by question mark (?).

PATH PARAMETER: is a part of url and added to the end of the url. Usually record id is used as path parameters to retrieve a specific record from database with GET request:
Note: Record ID is a unique ID for each record added to database. It is generated by system

/ (Path Parameter)

- **Used to identify a specific resource** or a collection of resources.
- Appears **before the question mark (?)** in the URL.

Example:

https://gorest.co.in/public/v1/users/123

This URL fetches **user with ID 123**.

Another Example:

https://api.example.com/products/45/reviews

- Fetches **reviews for product ID 45**.

? (Query Parameter)

- **Used to filter, search, or customize the response**.
- Appears **after the path** in the URL.
- Multiple query parameters are joined using &.

Example:

```
https://gorest.co.in/public/v1/users?page=2&status=active
```

- This URL **filters** active users on **page 2**.

Another Example:

`https://api.example.com/products?category=shoes&sort=price`

- Fetches **shoes** and **sorts** by **price**.

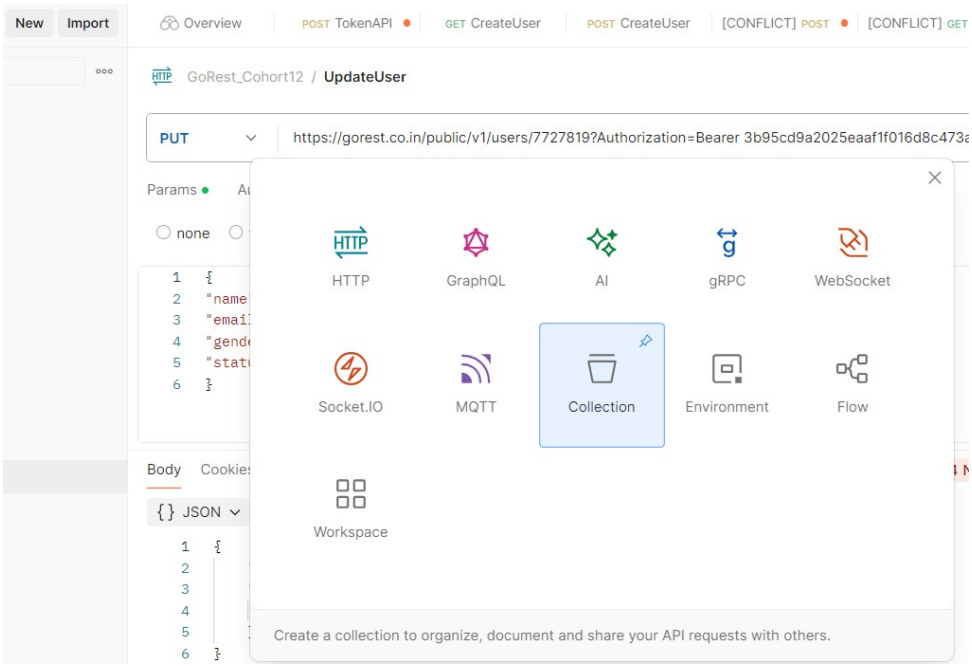
Collections

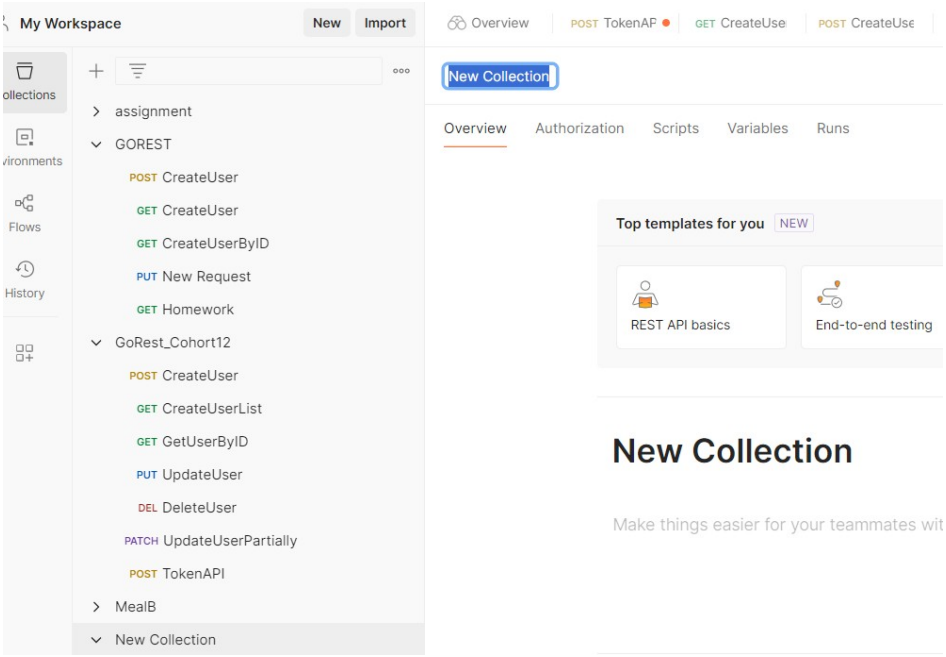
The Term Collections refers to a group of related API requests that are organized together for easier management and execution.

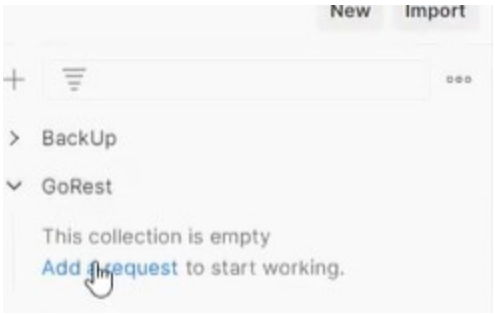
Click on New / Collection

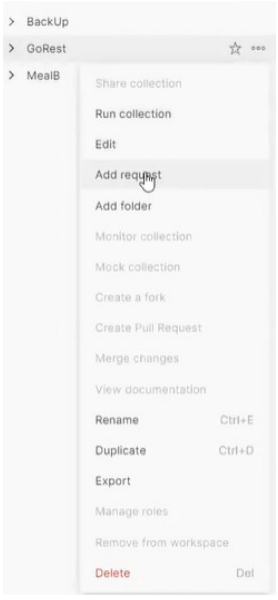
Name it

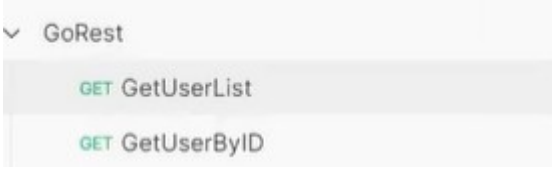
Add requests / or click on the three …



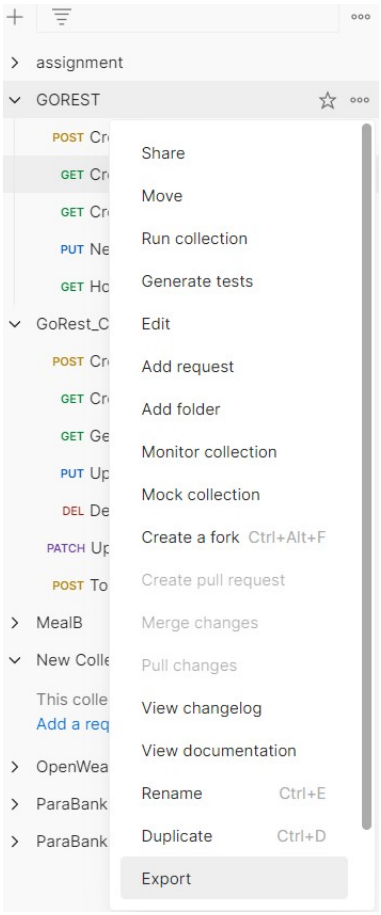

My Workspace
New
Import
Overview
POST TokenAP
GET CreateUse
POST CreateUse
New Collection
Overview
Authorization
Scripts
Variables
Runs
assignment
GOREST
POST CreateUser
GET CreateUser
GET CreateUserByID
PUT New Request
GET Homework
GoRest_Cohort12
POST CreateUser
GET CreateUserList
GET GetUserByID
PUT UpdateUser
DEL DeleteUser
PATCH UpdateUserPartially
POST TokenAPI
MealB
New Collection
Top templates for you
NEW
REST API basics
End-to-end testing
New Collection
Make things easier for your teammates wit

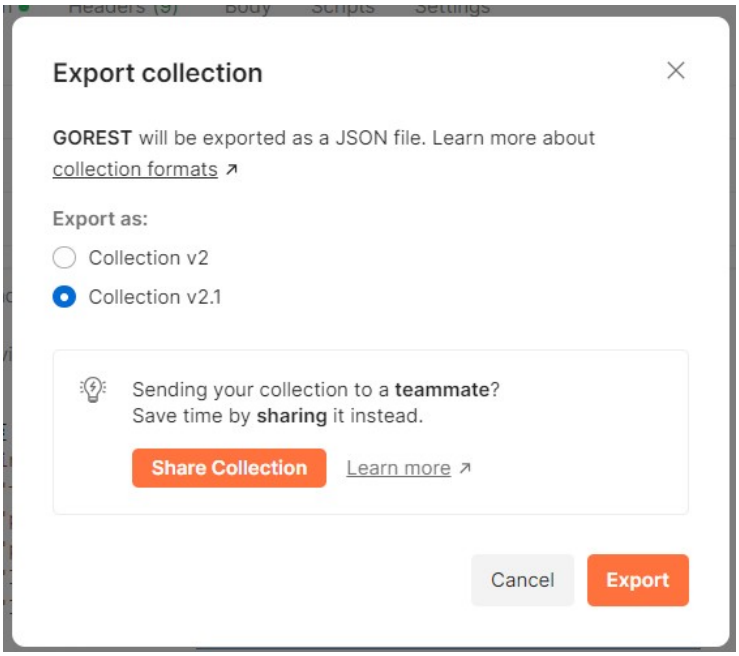

New
Import
BackUp
GoRest
This collection is empty
Add a request to start working.

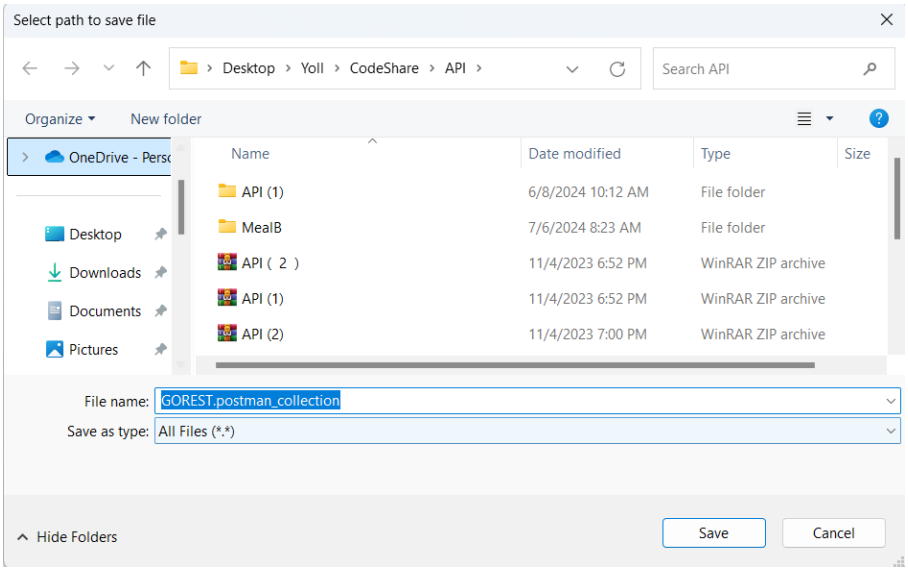

BackUp
GoRest
MealB
Share collection
Run collection
Edit
Add request
Add folder
Monitor collection
Mock collection
Create a fork
Create Pull Request
Merge changes
View documentation
Rename Ctrl+E
Duplicate Ctrl+D
Export
Manage roles
Remove from workspace
Delete Del


GoRest
GET GetUserList
GET GetUserByID

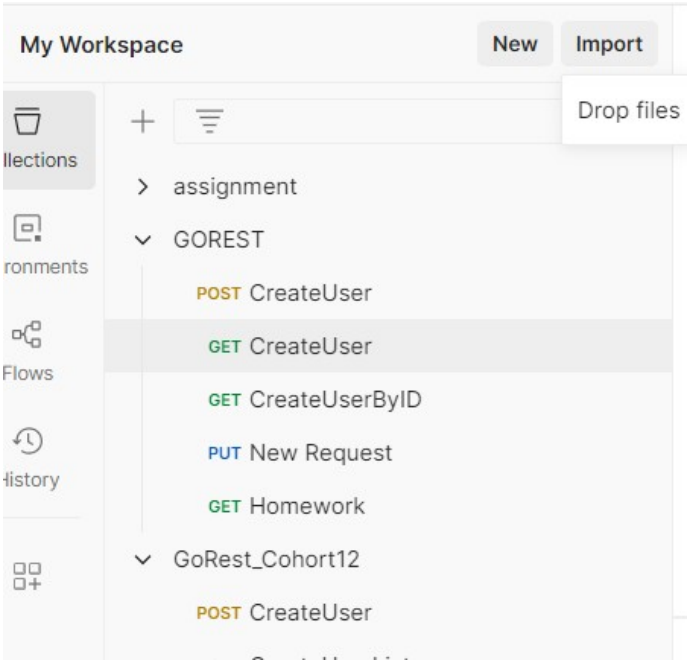
Export a collection :

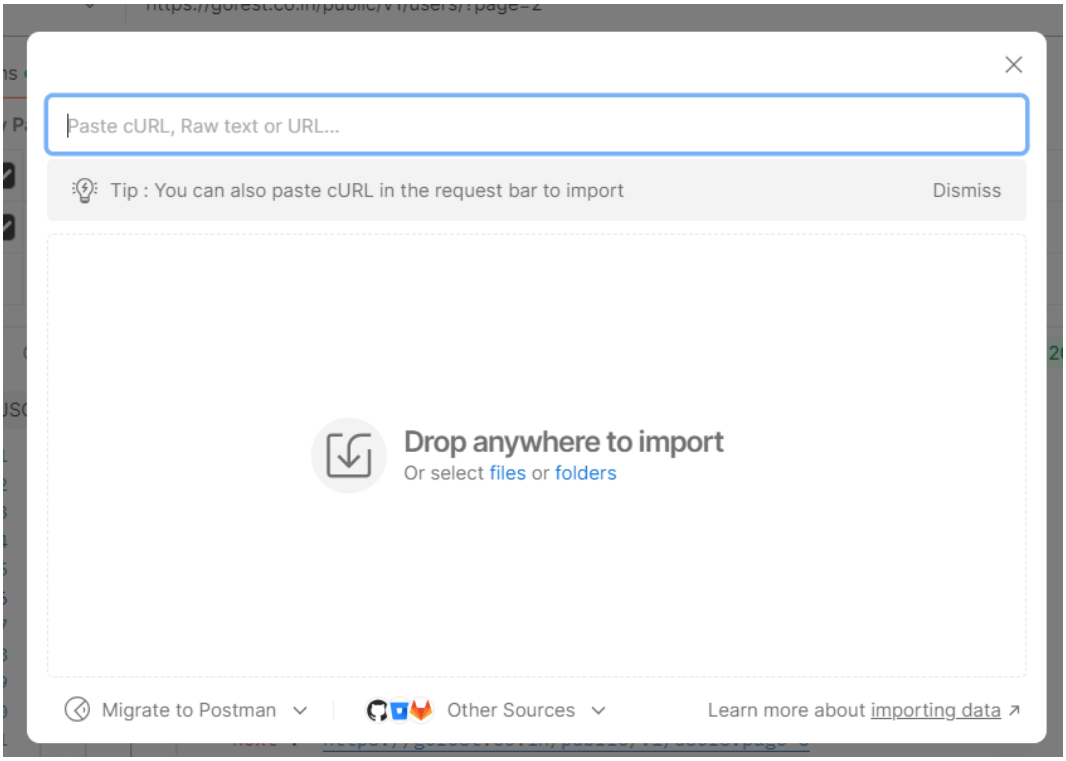


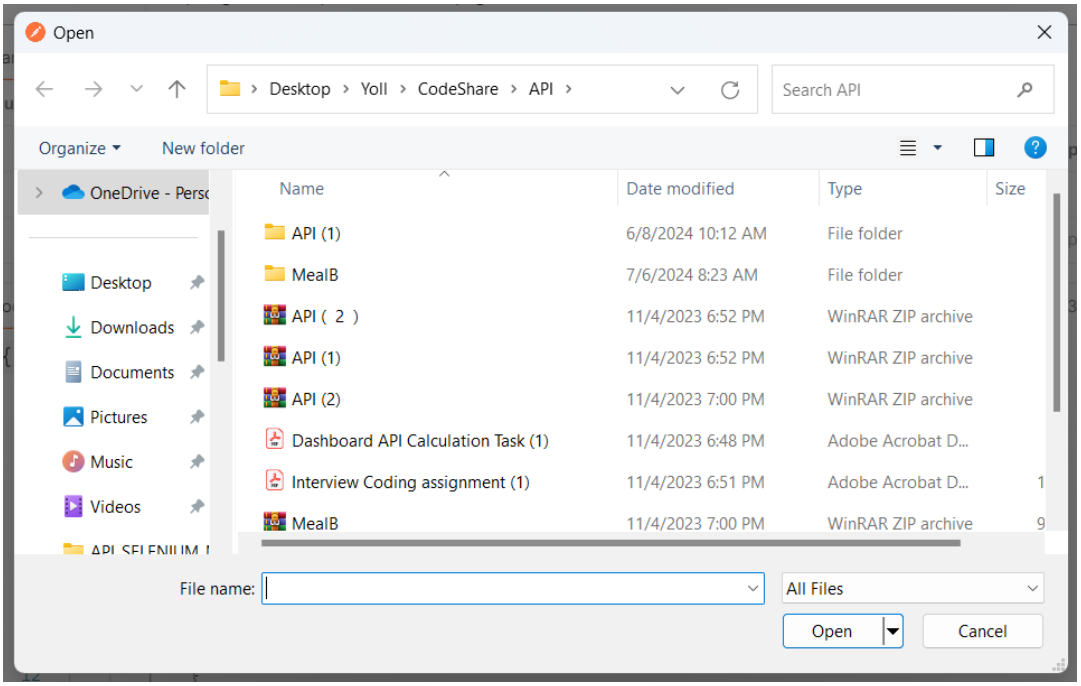


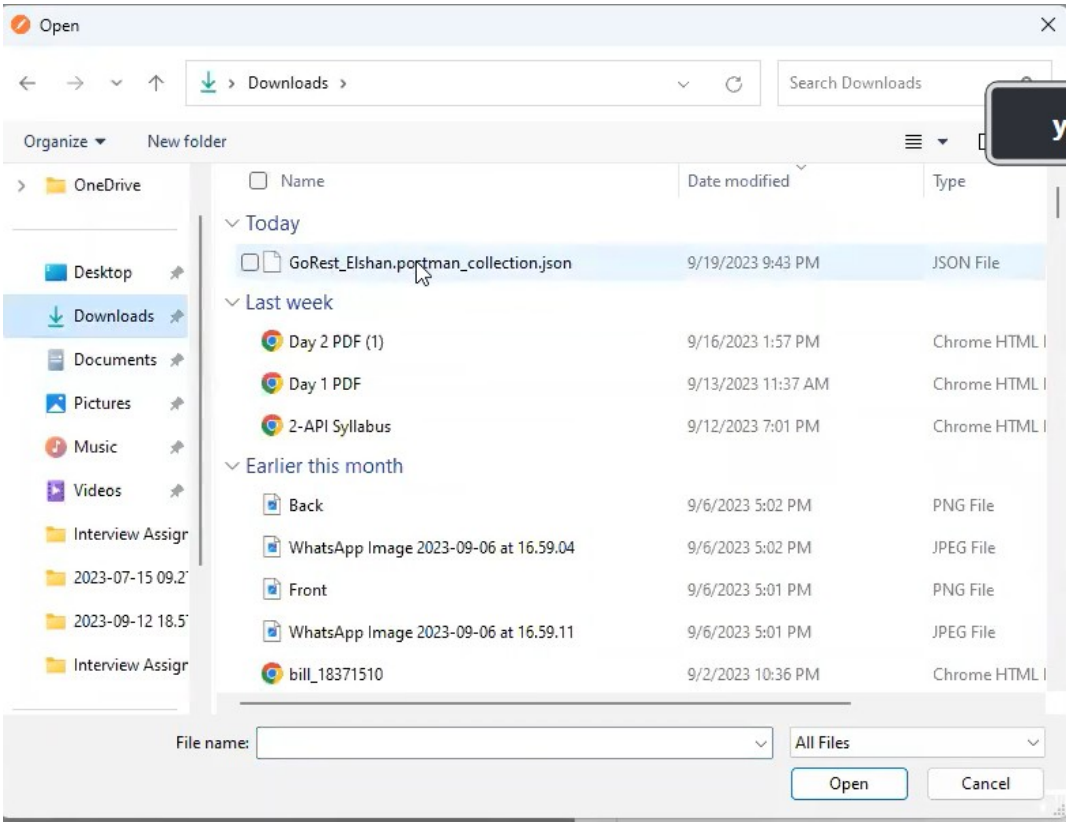

Select path to save file
Desktop › Yoll › CodeShare › API ›
Search API
Organize
New folder
OneDrive - Perso
Desktop
Downloads
Documents
Pictures
Name
Date modified
Type
Size
API (1)
6/8/2024 10:12 AM
File folder
MealB
7/6/2024 8:23 AM
File folder
API (2)
11/4/2023 6:52 PM
WinRAR ZIP archive
API (1)
11/4/2023 6:52 PM
WinRAR ZIP archive
API (2)
11/4/2023 7:00 PM
WinRAR ZIP archive
File name:
GOREST.postman_collection
Save as type:
All Files (*.*)
Hide Folders
Save
Cancel

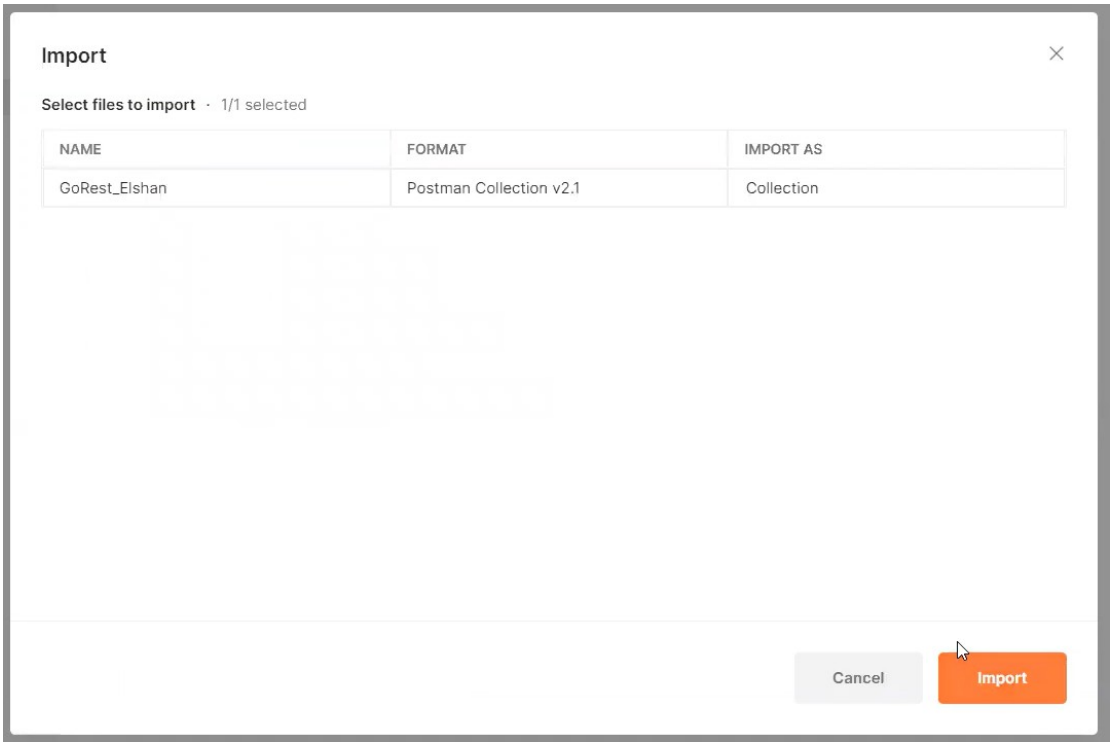
Import


My Workspace
New
Import
Drop files
llections
ronments
Flows
istory
assignment
GOREST
POST CreateUser
GET CreateUser
GET CreateUserByID
PUT New Request
GET Homework
GoRest_Cohort12
POST CreateUser


Paste cURL, Raw text or URL...
Tip : You can also paste cURL in the request bar to import
Dismiss
Drop anywhere to import
Or select files or folders
Migrate to Postman
Other Sources
Learn more about importing data


Open
Desktop › Yoll › CodeShare › API ›
Search API
Organize
New folder
OneDrive - Perso
Desktop
Downloads
Documents
Pictures
Music
Videos
Name
Date modified
Type
Size
API (1) 6/8/2024 10:12 AM File folder
MealB 7/6/2024 8:23 AM File folder
API (2) 11/4/2023 6:52 PM WinRAR ZIP archive
API (1) 11/4/2023 6:52 PM WinRAR ZIP archive
API (2) 11/4/2023 7:00 PM WinRAR ZIP archive
Dashboard API Calculation Task (1) 11/4/2023 6:48 PM Adobe Acrobat D...
Interview Coding assignment (1) 11/4/2023 6:51 PM Adobe Acrobat D...
MealB 11/4/2023 7:00 PM WinRAR ZIP archive
File name:
All Files
Open
Cancel


Open
Downloads
Search Downloads
Organize
New folder
OneDrive
Desktop
Downloads
Documents
Pictures
Music
Videos
Interview Assigr
2023-07-15 09.2
2023-09-12 18.5
Interview Assigr
Name
Date modified
Type
Today
GoRest_Elshan.postman_collection.json
9/19/2023 9:43 PM
JSON File
Last week
Day 2 PDF (1)
9/16/2023 1:57 PM
Chrome HTML
Day 1 PDF
9/13/2023 11:37 AM
Chrome HTML
2-API Syllabus
9/12/2023 7:01 PM
Chrome HTML
Earlier this month
Back
9/6/2023 5:02 PM
PNG File
WhatsApp Image 2023-09-06 at 16.59.04
9/6/2023 5:02 PM
JPEG File
Front
9/6/2023 5:01 PM
PNG File
WhatsApp Image 2023-09-06 at 16.59.11
9/6/2023 5:01 PM
JPEG File
bill_18371510
9/2/2023 10:36 PM
Chrome HTML
File name:
All Files
Open
Cancel


Import
Select files to import · 1/1 selected
NAME
FORMAT
IMPORT AS
GoRest_Elshan
Postman Collection v2.1
Collection
Cancel
Import

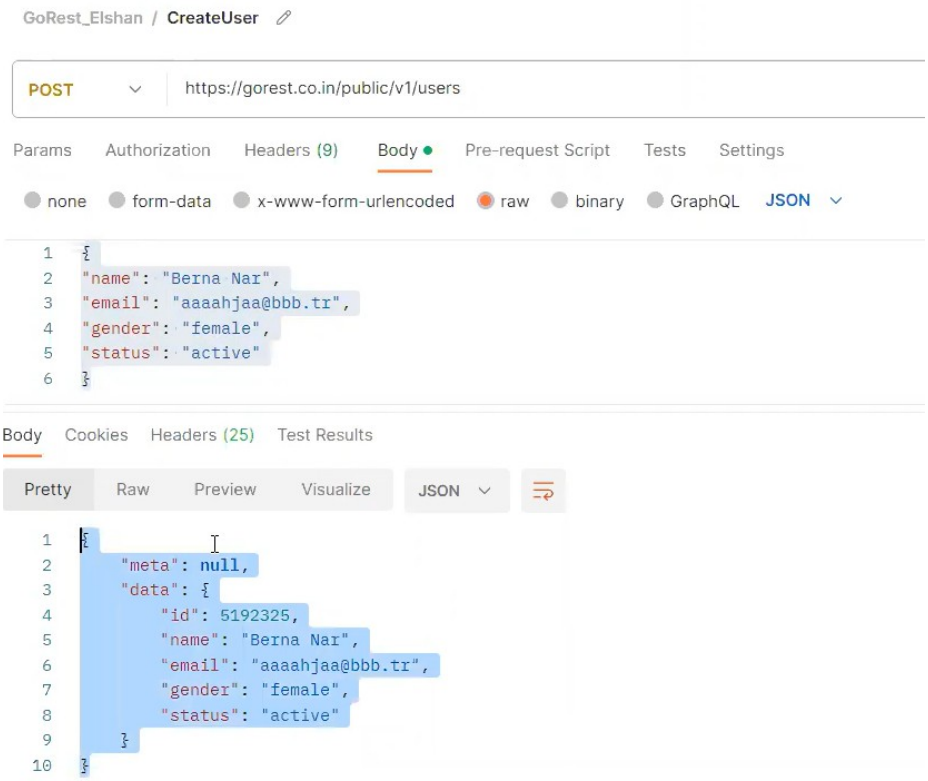
API Payload

API PAYLOAD

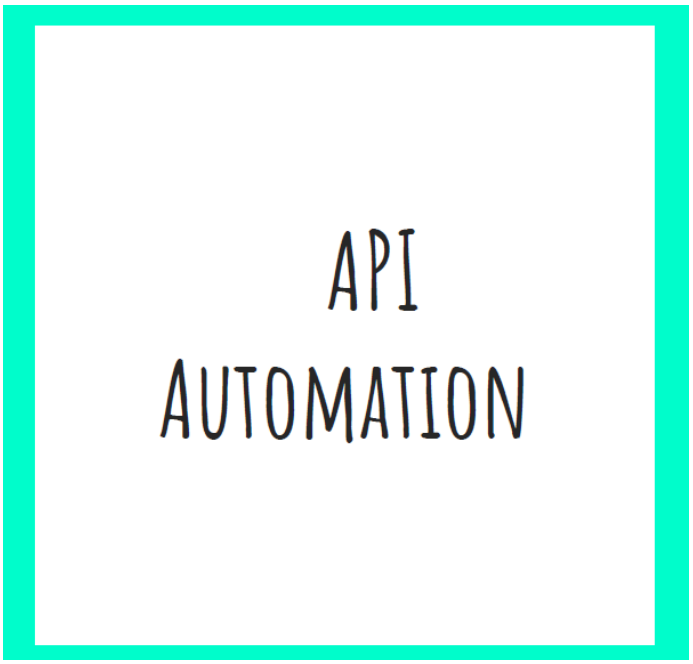
API Payload is the data:

- we submit in request
- we receive in response

Example of request payload and response Payload:

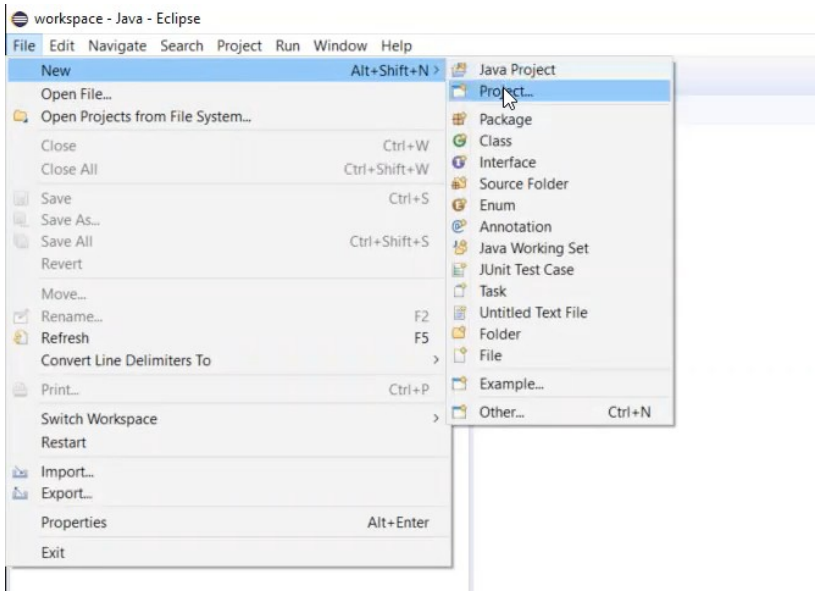


API AUTOMATION

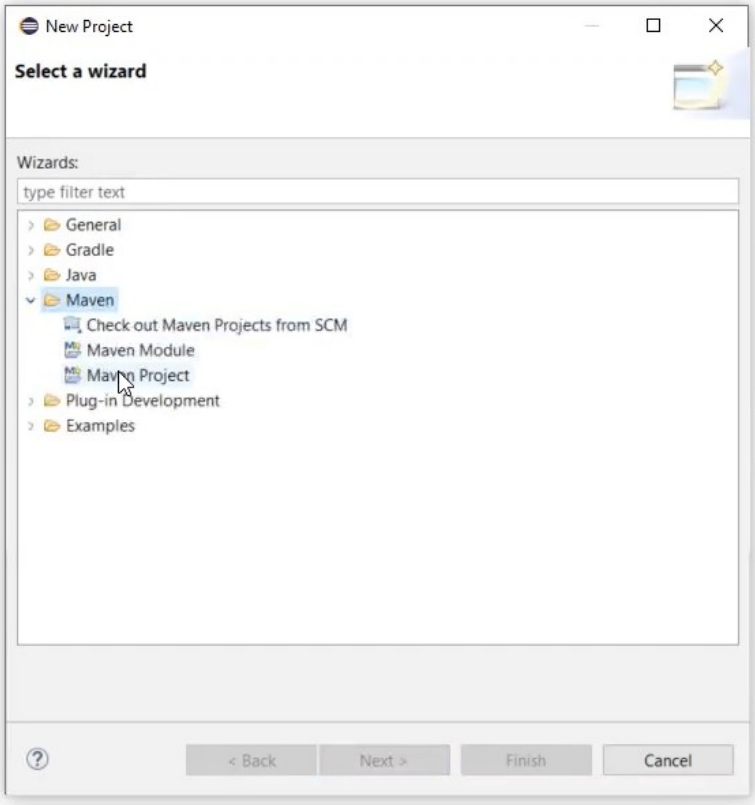


Create Maven file on Eclipse Software

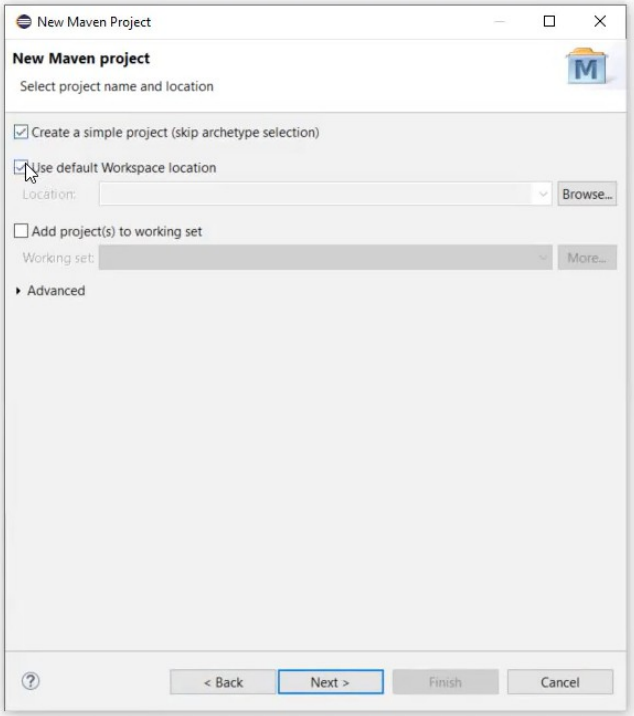
Go to File/New/ Select PROJECT not java Project/



Select Maven folder/ Select Maven Project



Next, and select : Crerate a simple project (skip archetype selection), and do not click on the third one : Add default project(s) to working set, then Click on Next.

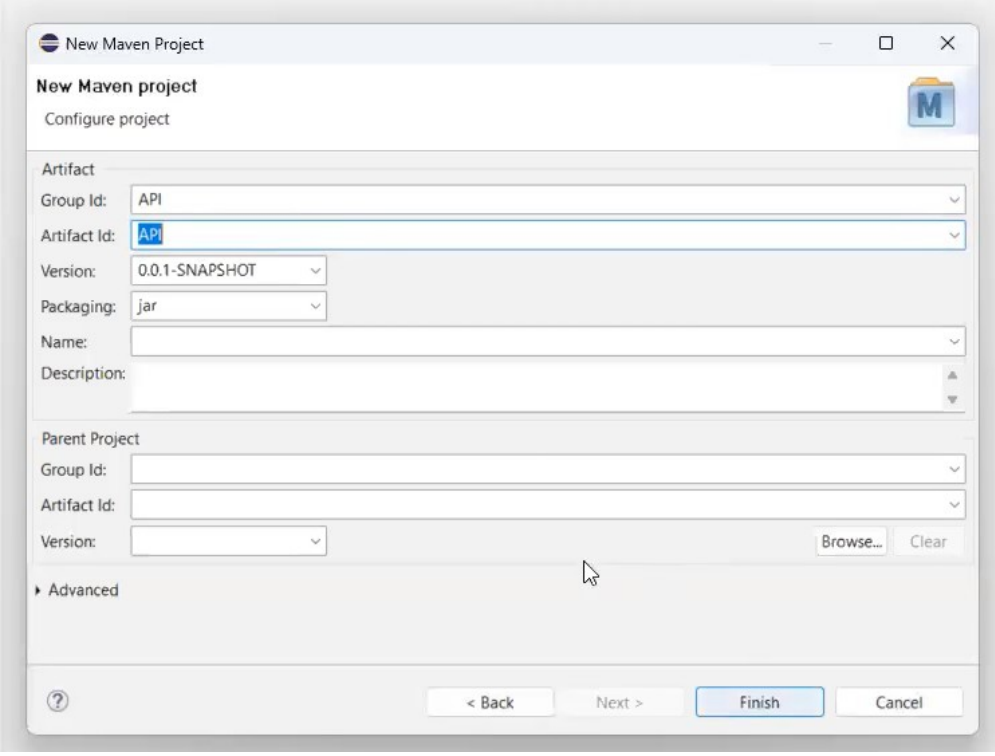


Write :

Group ID: API

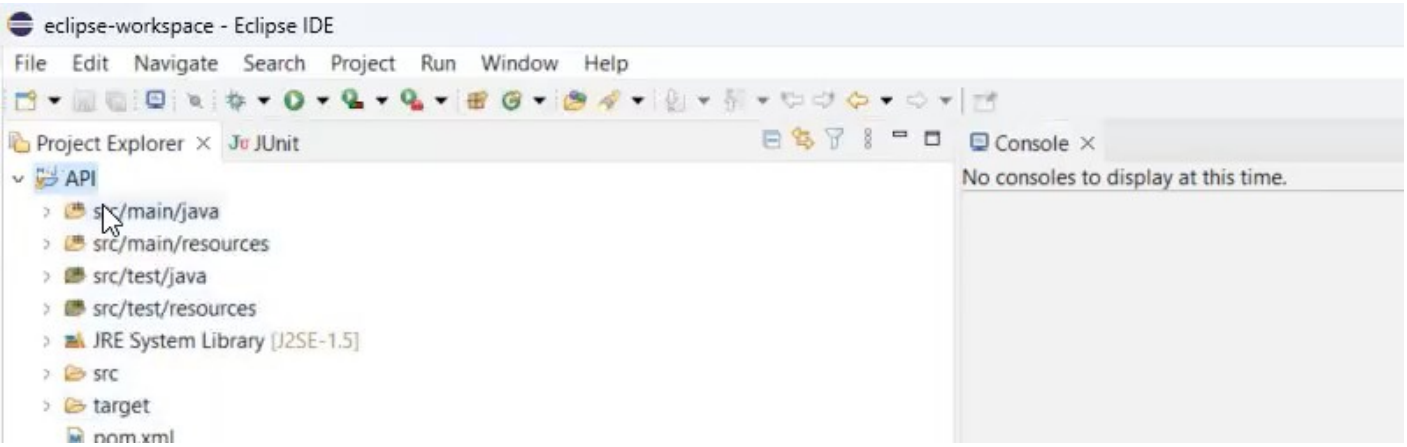
Artifact Id: API

Click on Finish Button



Result:

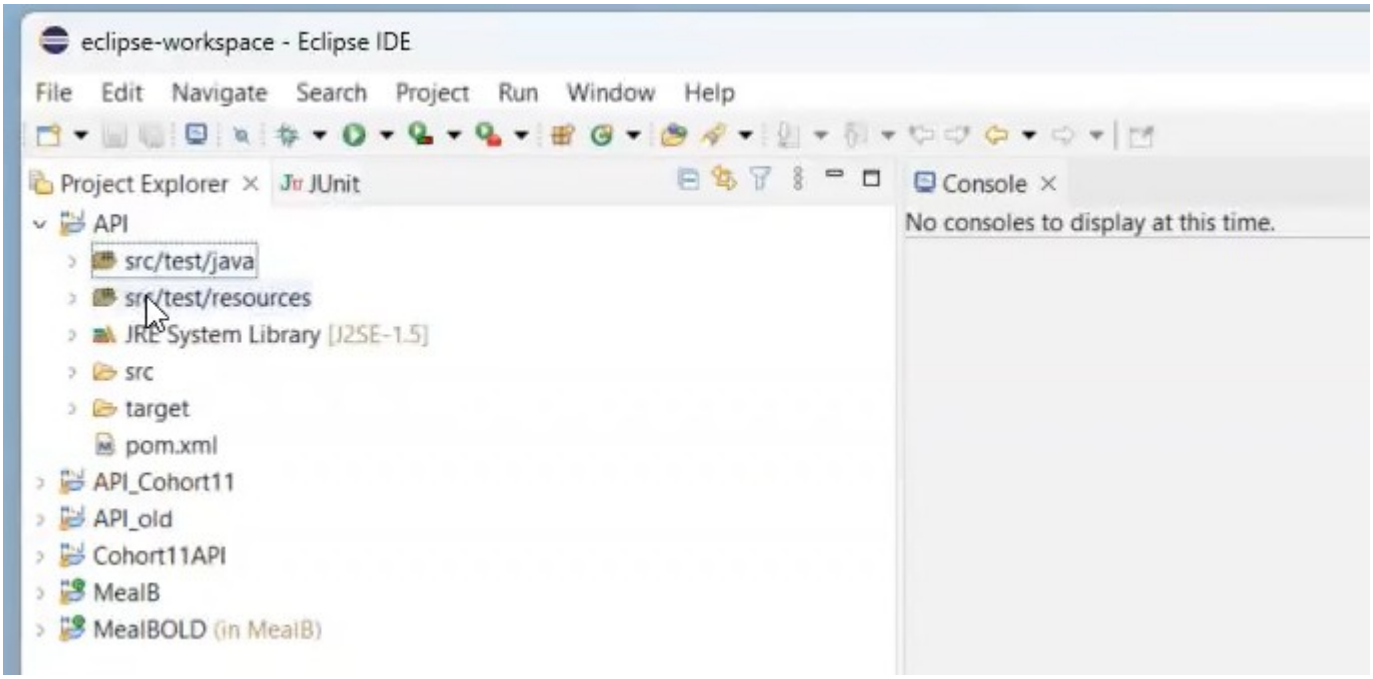
We can see the name of the Project: **API**



As Testers we use only :

src/test/java (**it contains ONLY Java Code**), and **src/test/ressources (it contains xml, json file, excel...NO Java Class)**

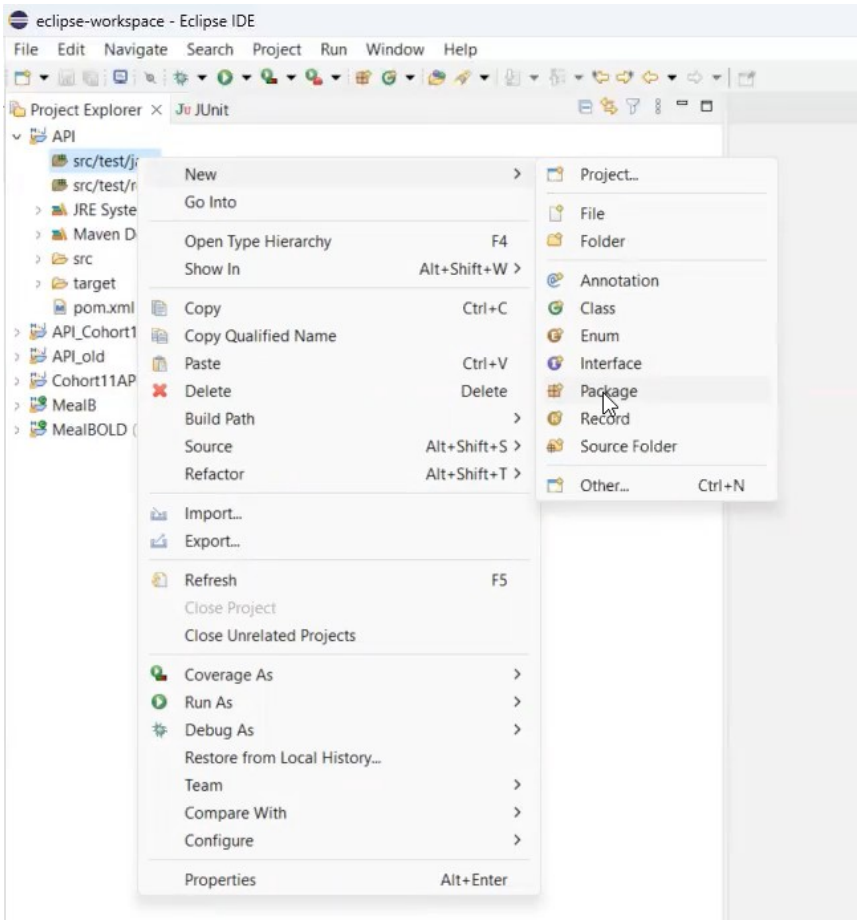
We do not to use : src/main/java and src/main/ressources, We delete them.



Create Packages under : src/test/java

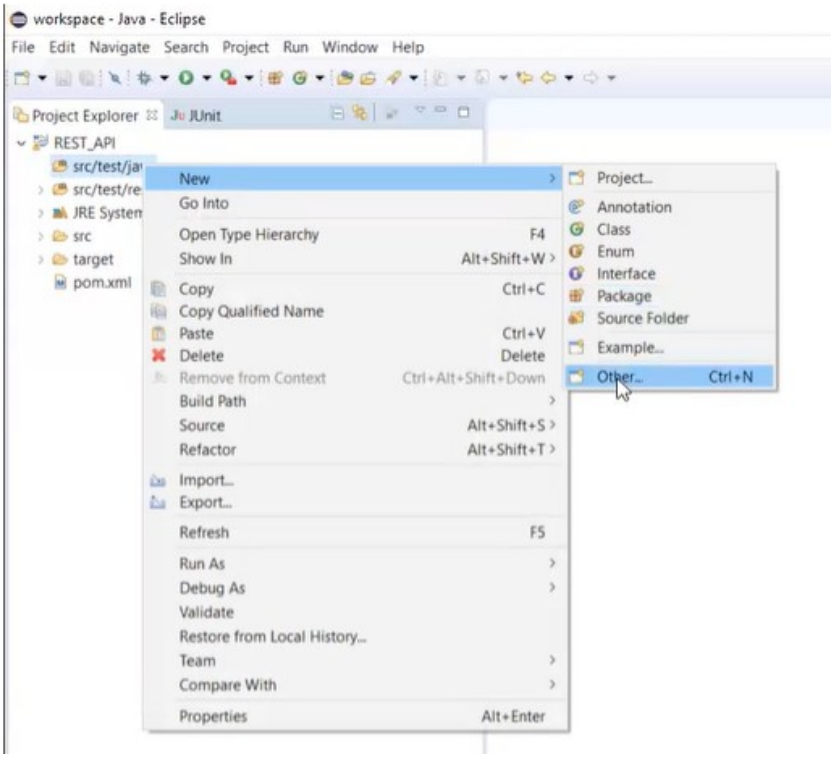
Cilck right on it**/ new / Package**

Name it : **gorest**

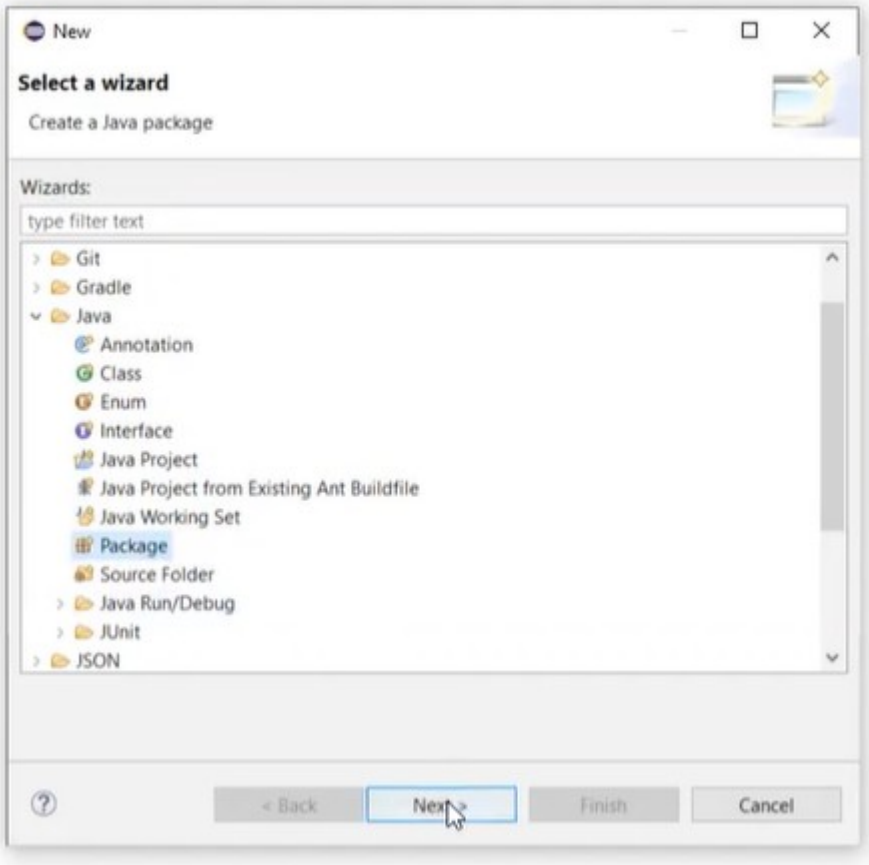


If you are unable to see Package (I used a different name of the package as an example)

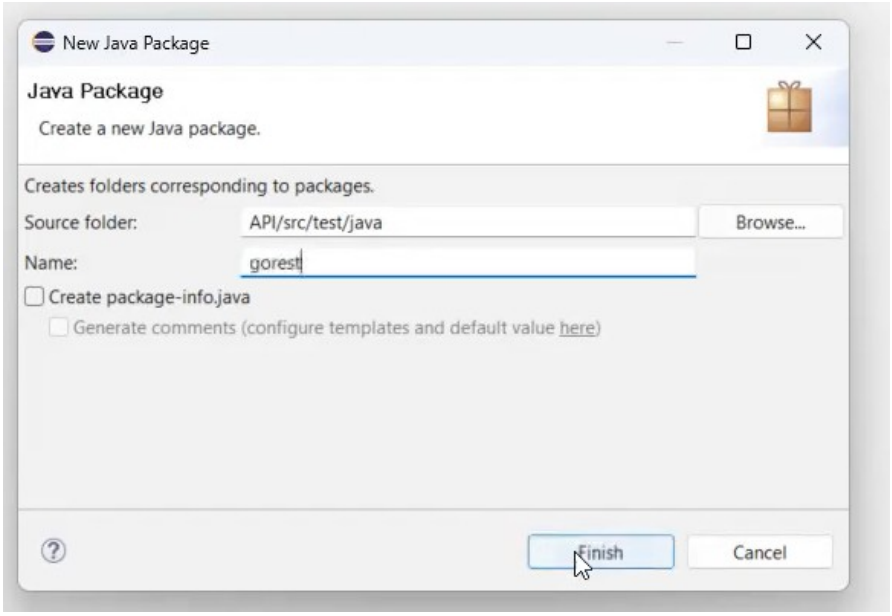
Go to Other:



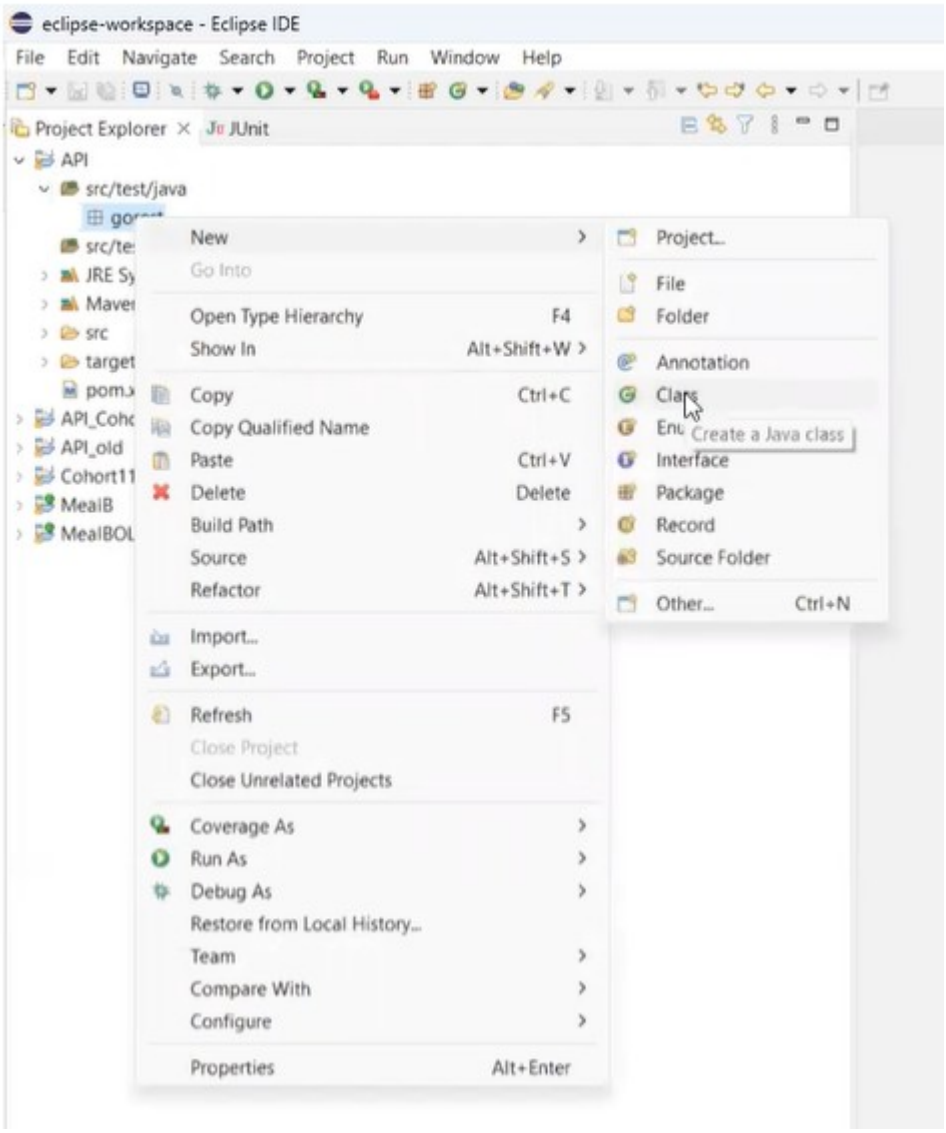
Choose Package under Java file



Put a name: gorest. Click on Finish Button



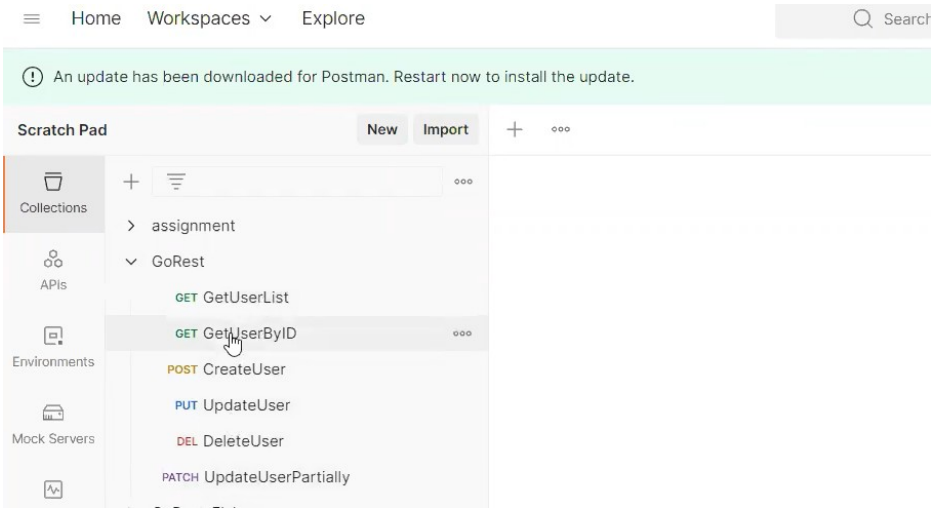
the gorest package is under : src/test/java

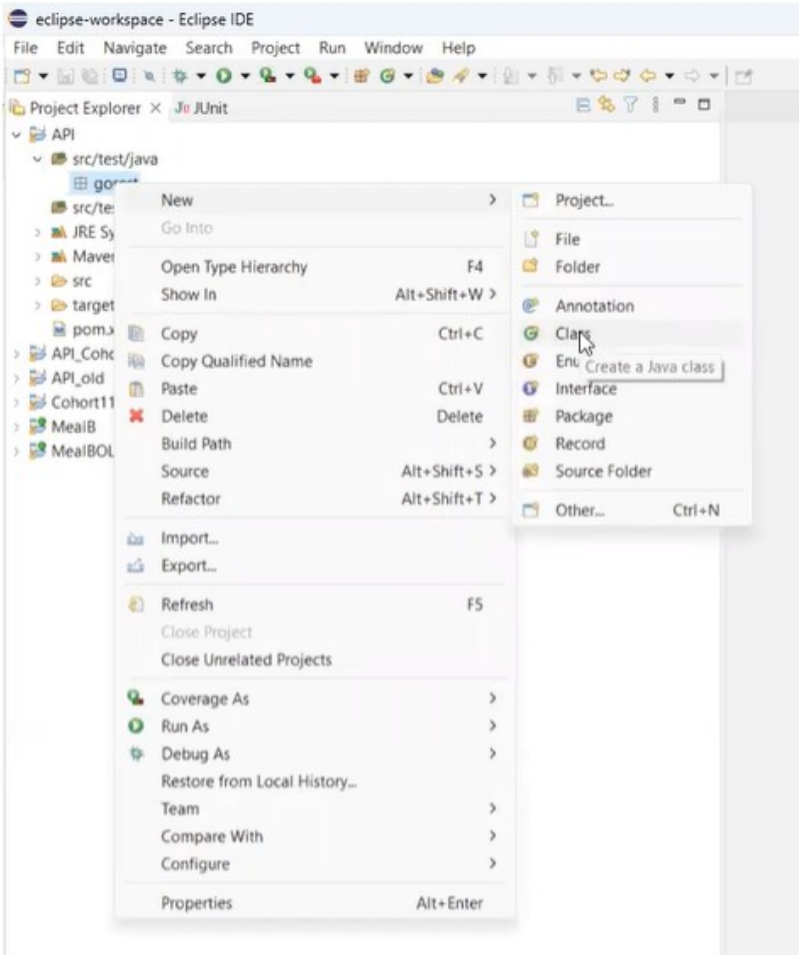


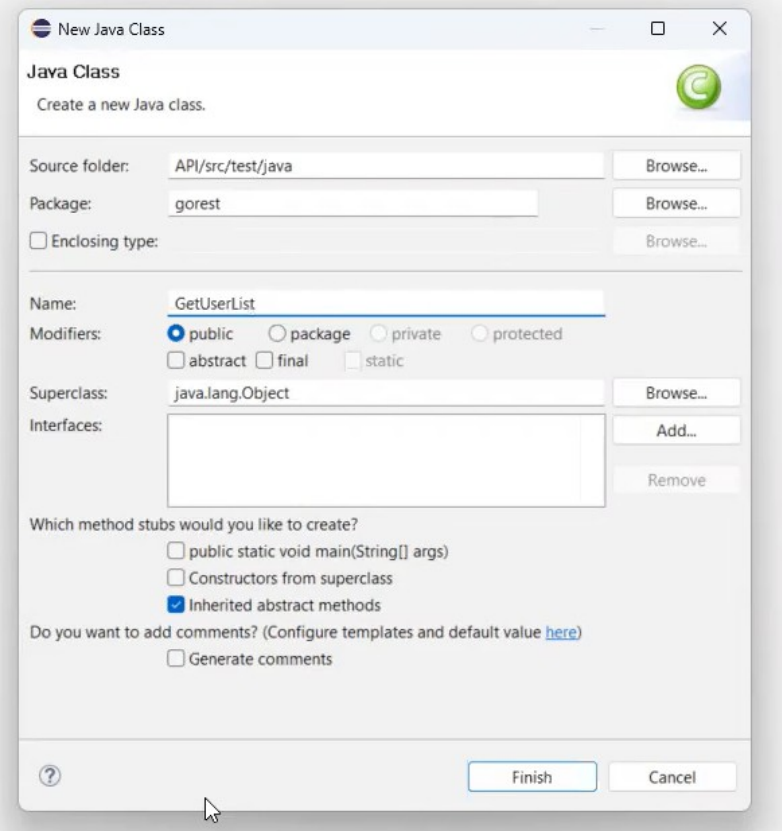
Then create a class under gorest Package:

Right Cilck on gorest package / new / class

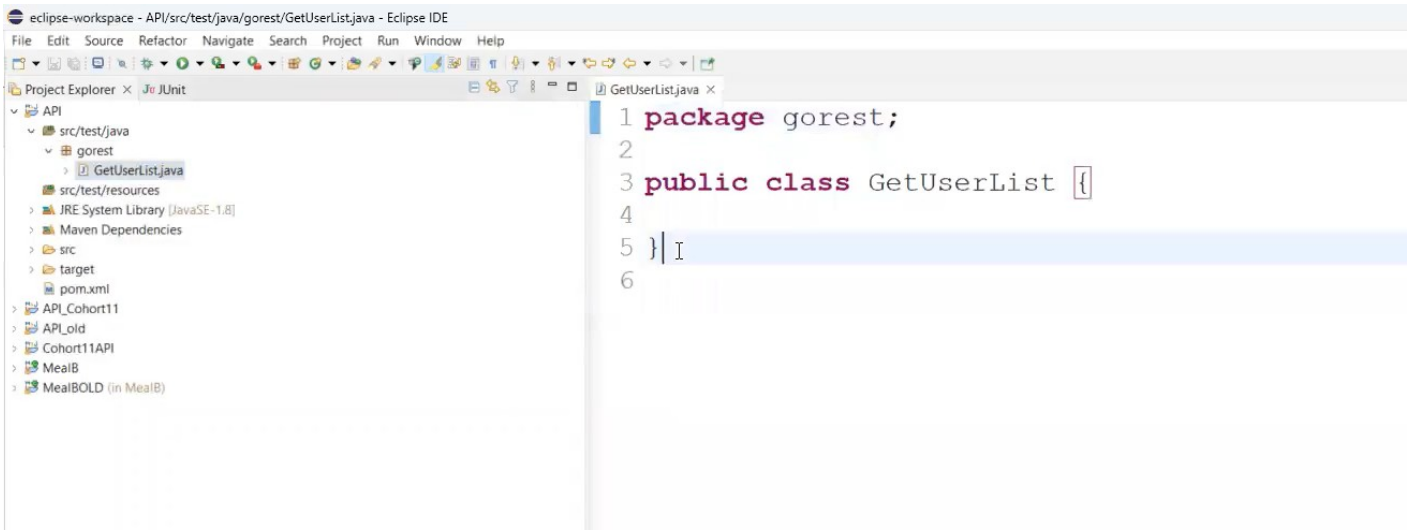
Name the Java Class : **GetUserList** as the Postman Request (**Get GetUserList**...)







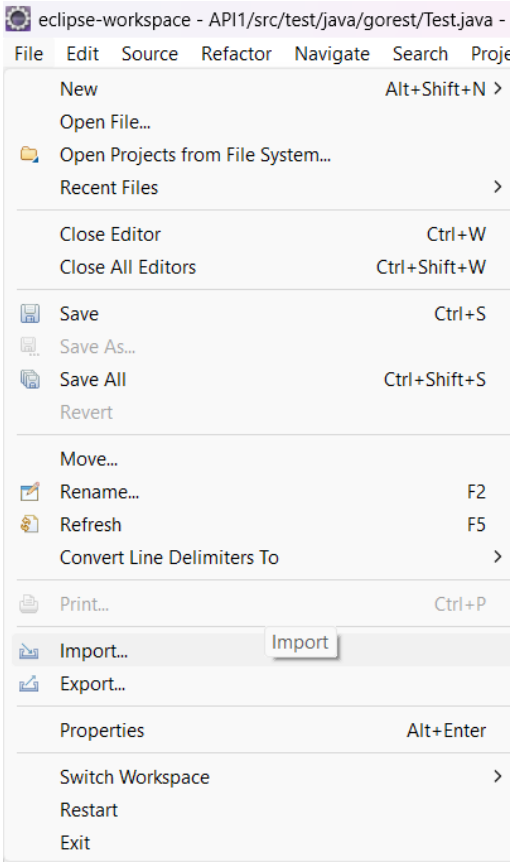
GetUserList Class



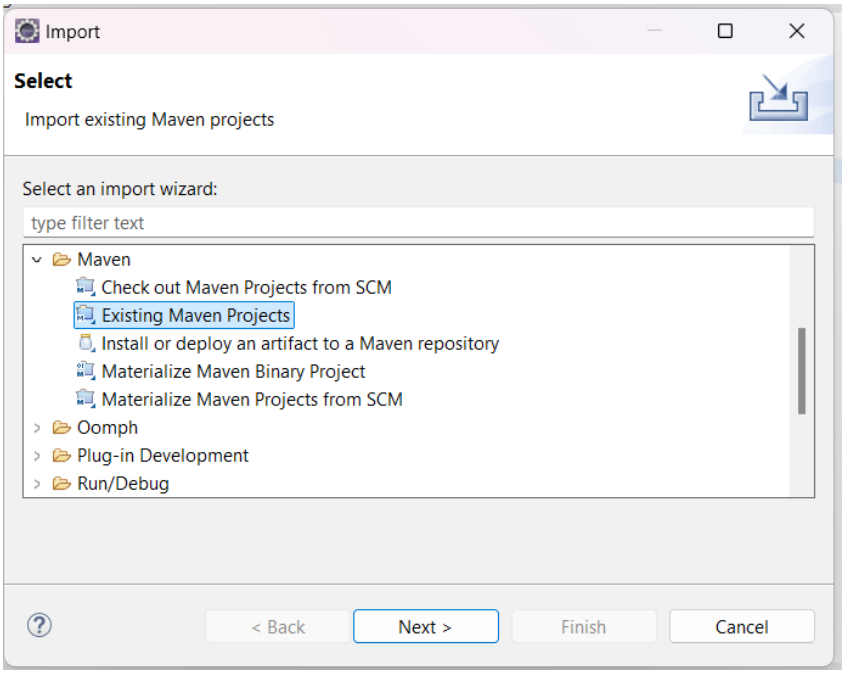
Each Java class must have the same name of the Requests on Postman !

How to upload a project created in Maven ?

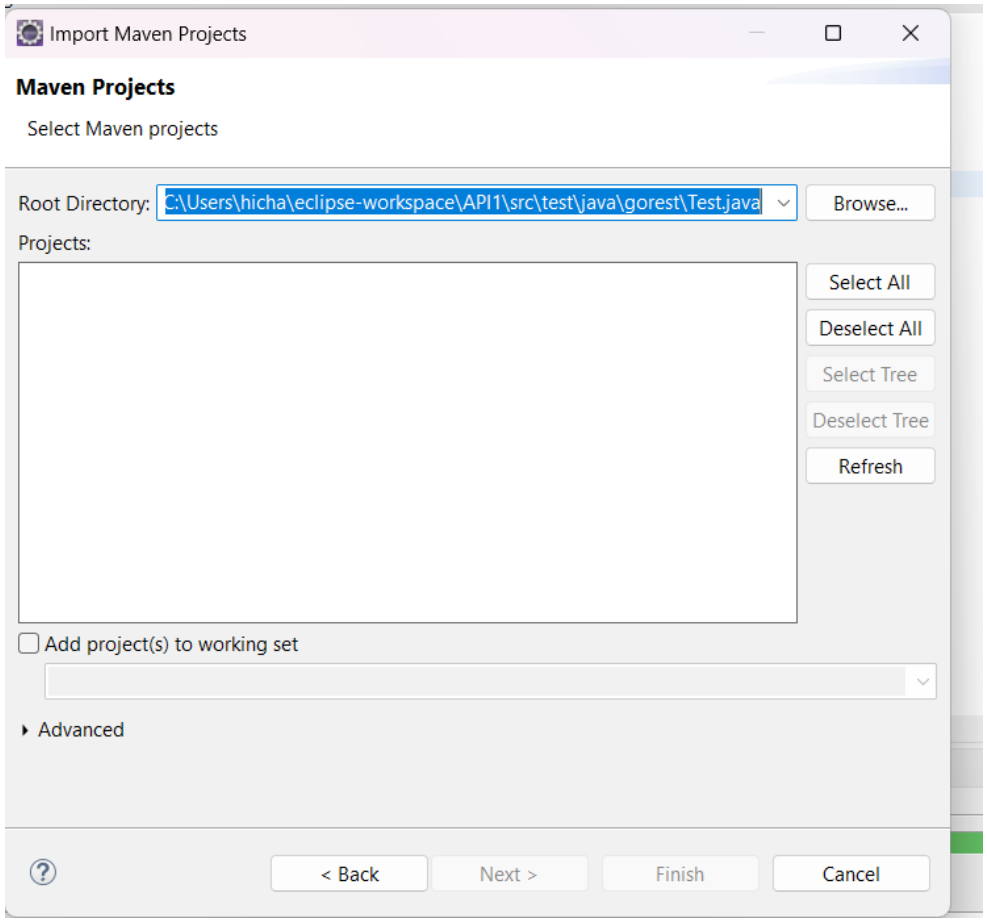
File / Import /

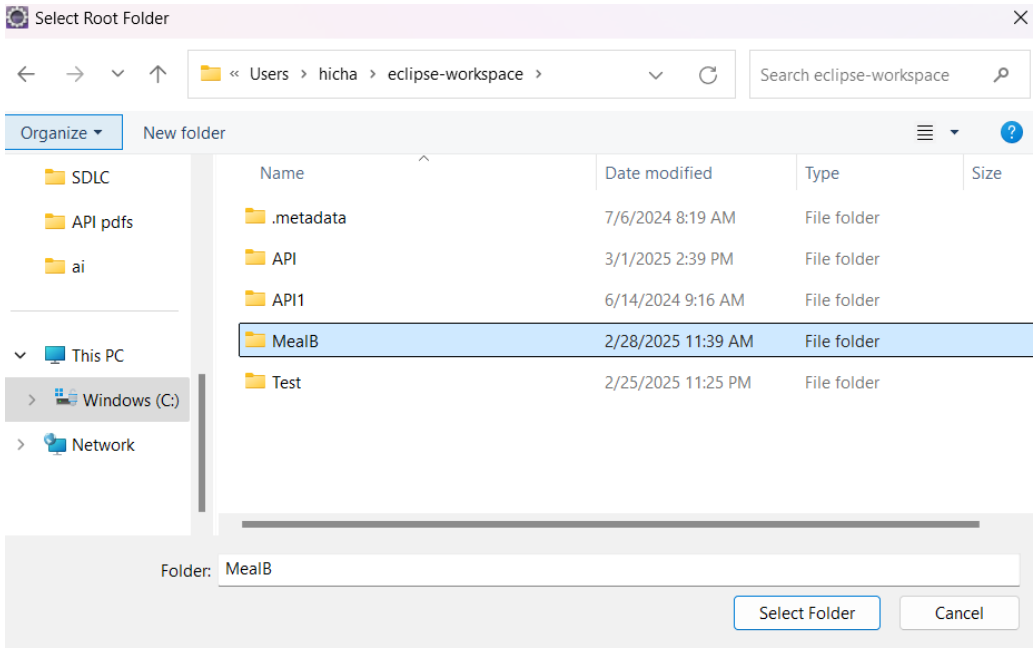


Maven / Existing Maven Projects / Next

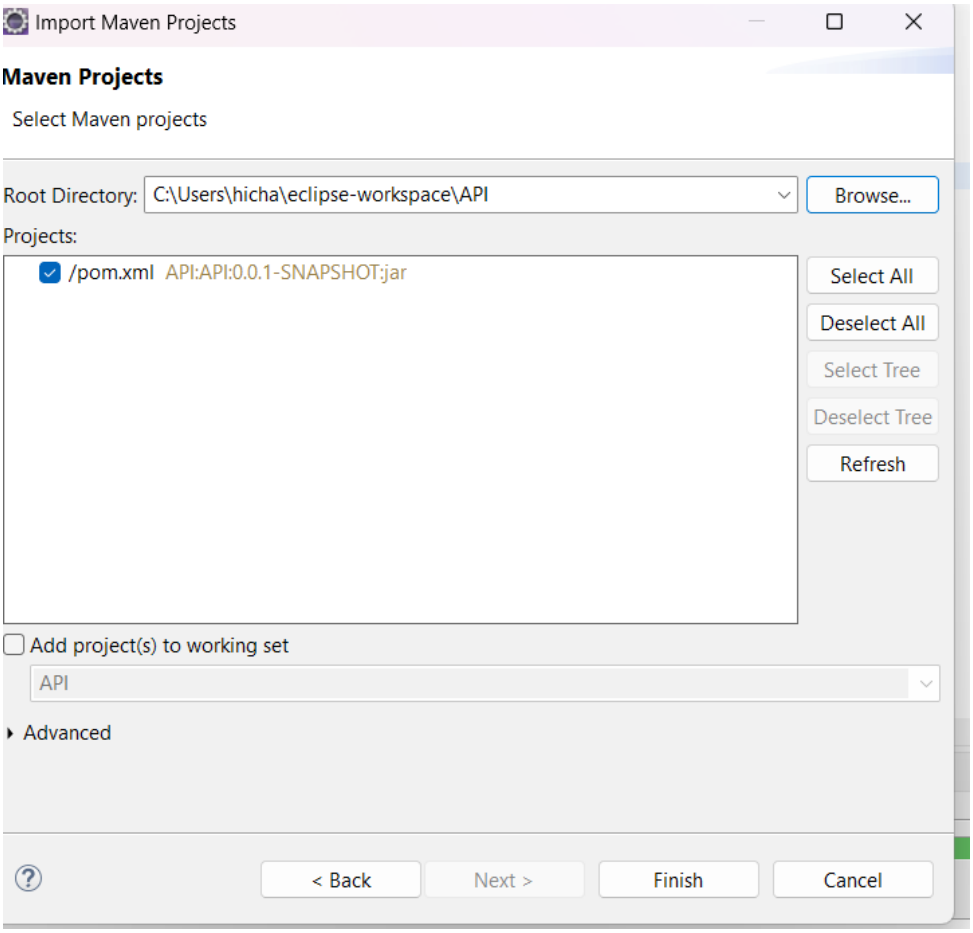


Browse / Select a folder





Select the jar file



REST ASSURED

Rest Assured is one of the most popular libraries used for Rest API Automation.

REST-assured

DEPENDENCY

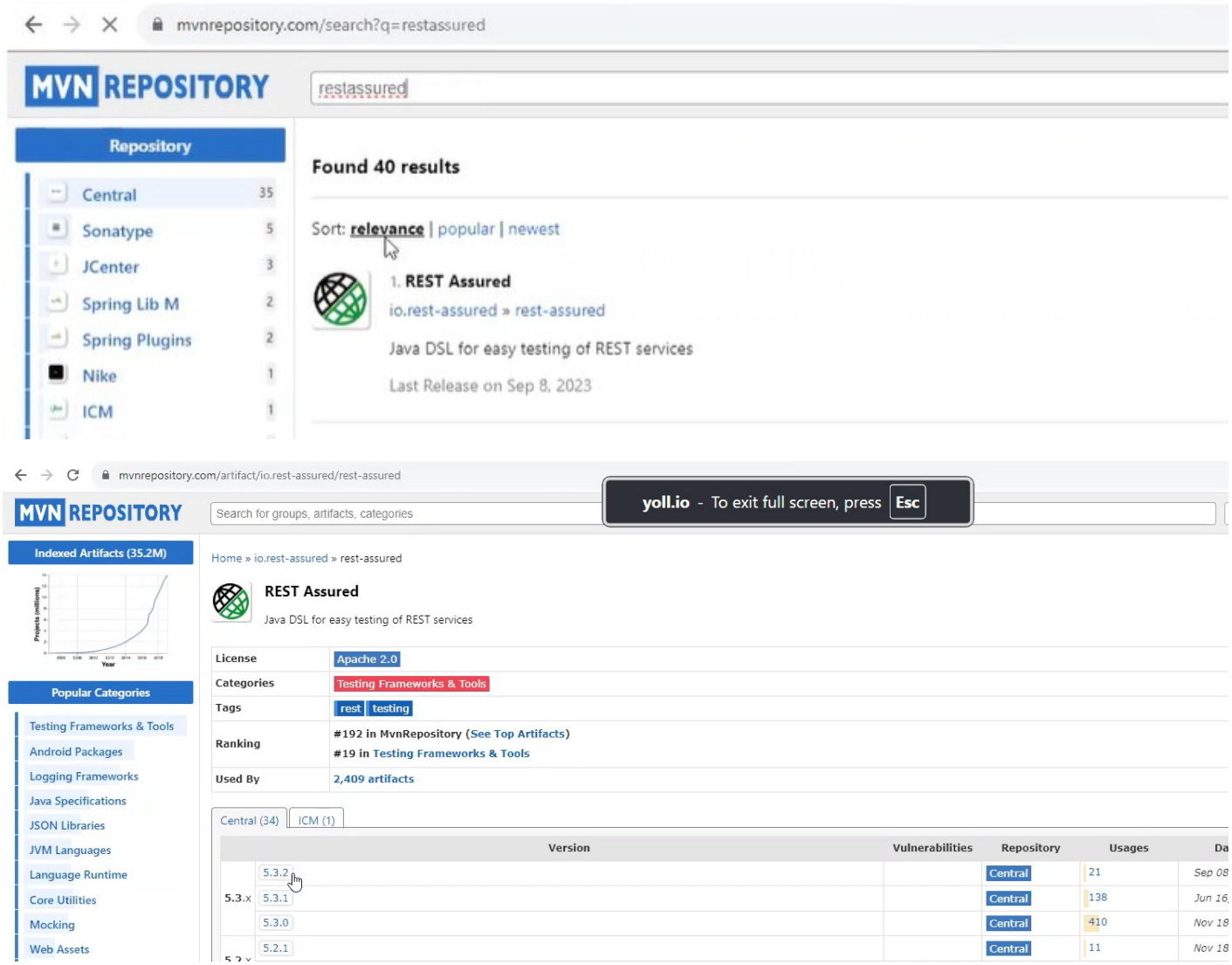
In order to Automate API scenarios, we need Rest Assured library.

Adding Dependencies on pom.xml file :

>>you need to get the all Dependencies from :

https://mvnrepository.com/

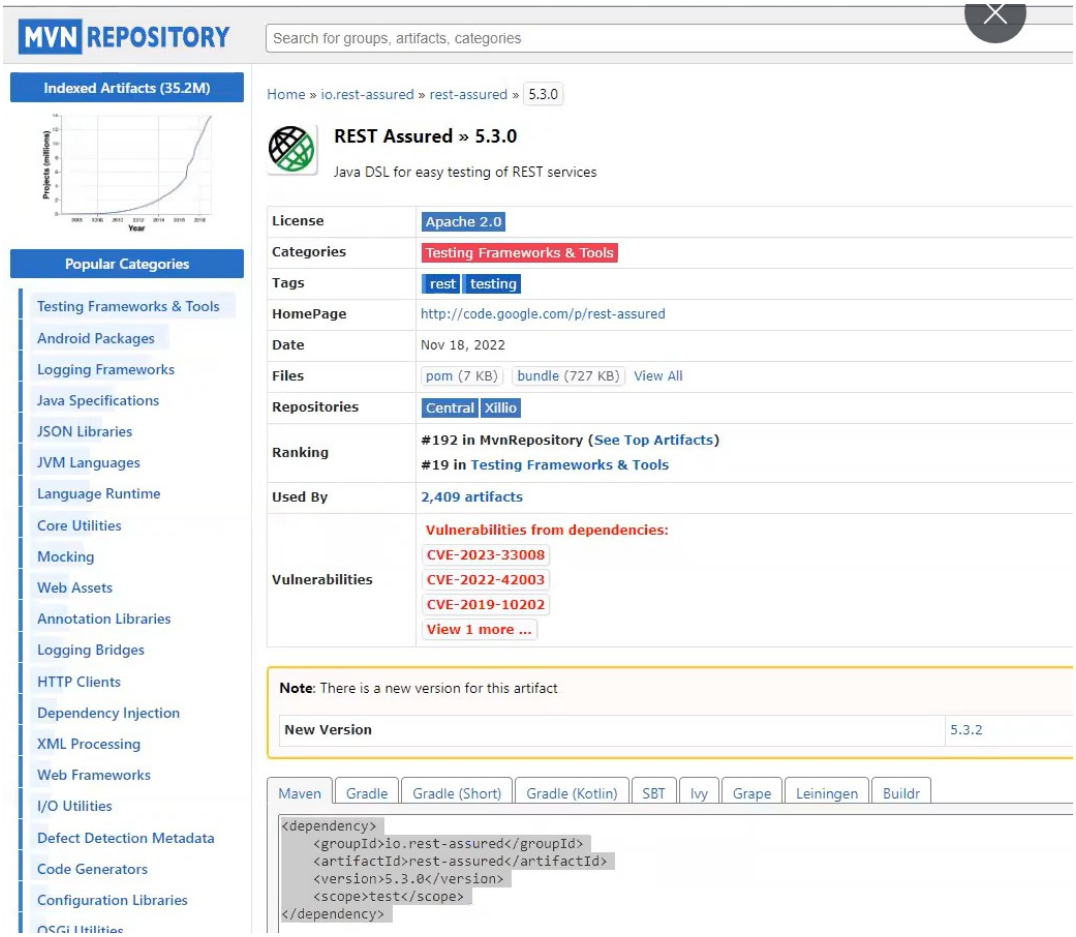
restassured



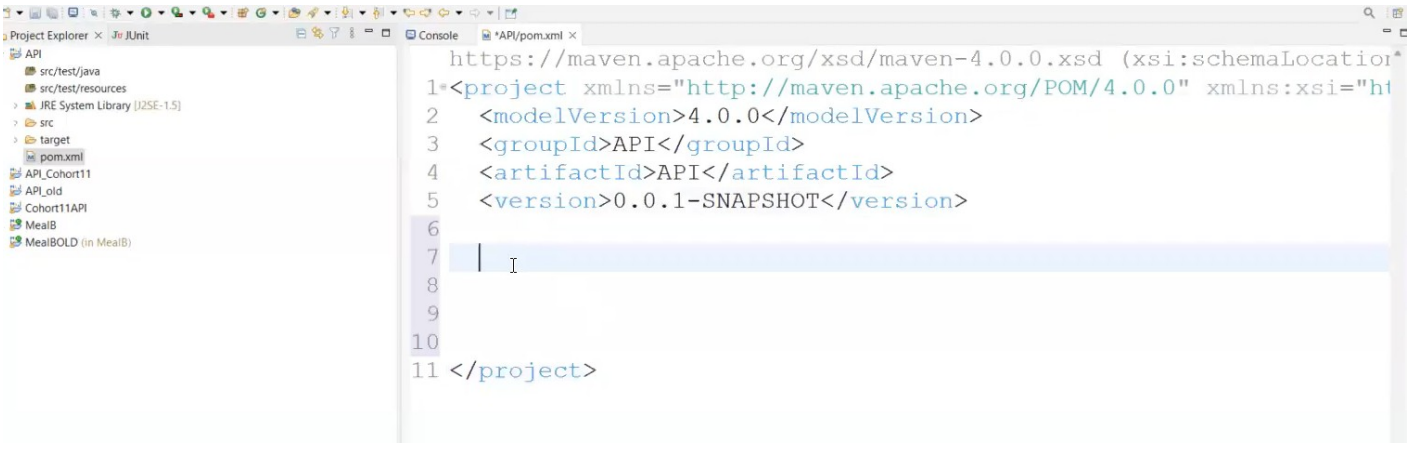
Choose the most used :410

Copy and paste om pom.xml

Use crtl +shift+f to beautify the code



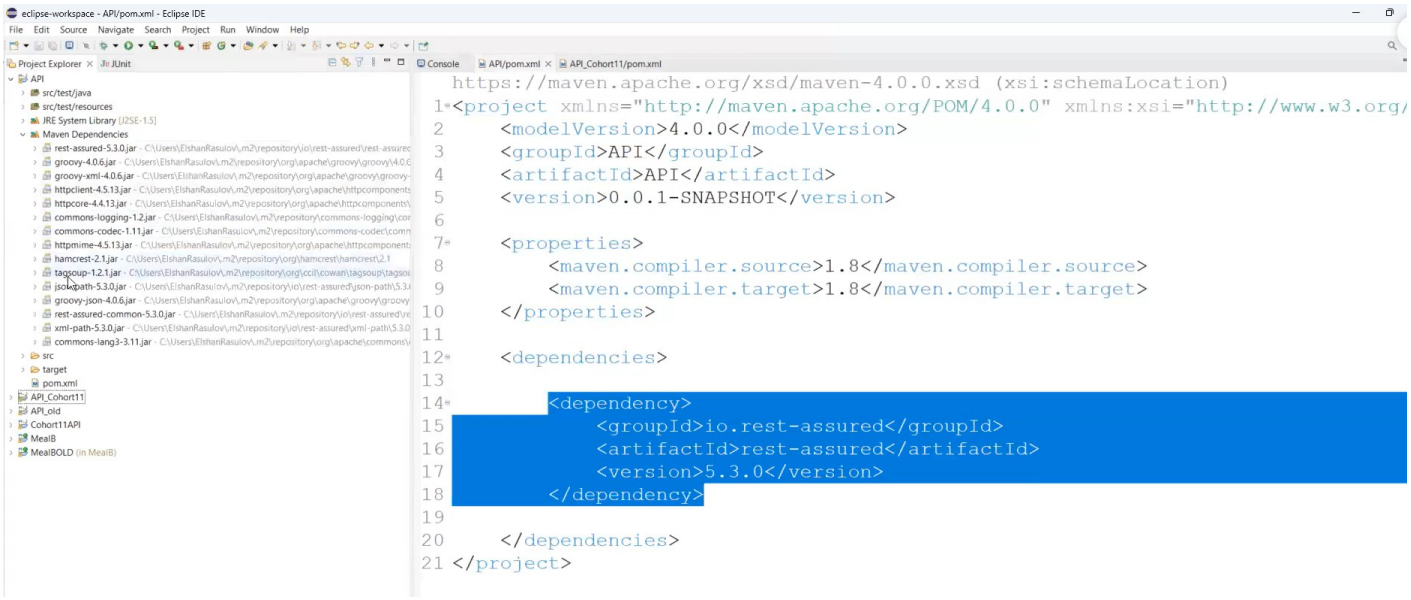
Click on pom.xml file, and create an extra space between <verion> and </project>:



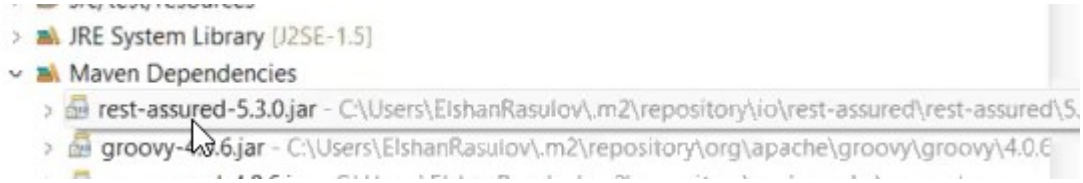
Paste the code below on the pom.xml

Put the dependencies between : <dependencies> and </dependencies> after </properties>

After Saving the file, you will see Maven Dependencies file on your left.



You can see the rest-assured file:



Add Junit Dependency:

The **JUnit dependency** is used in **Java** projects to facilitate **unit testing**. JUnit is a popular **testing framework** that allows developers to write and execute tests to verify the correctness of their code. It is commonly used in **test-driven development (TDD)** and for automating test cases in Java applications, including REST Assured tests.

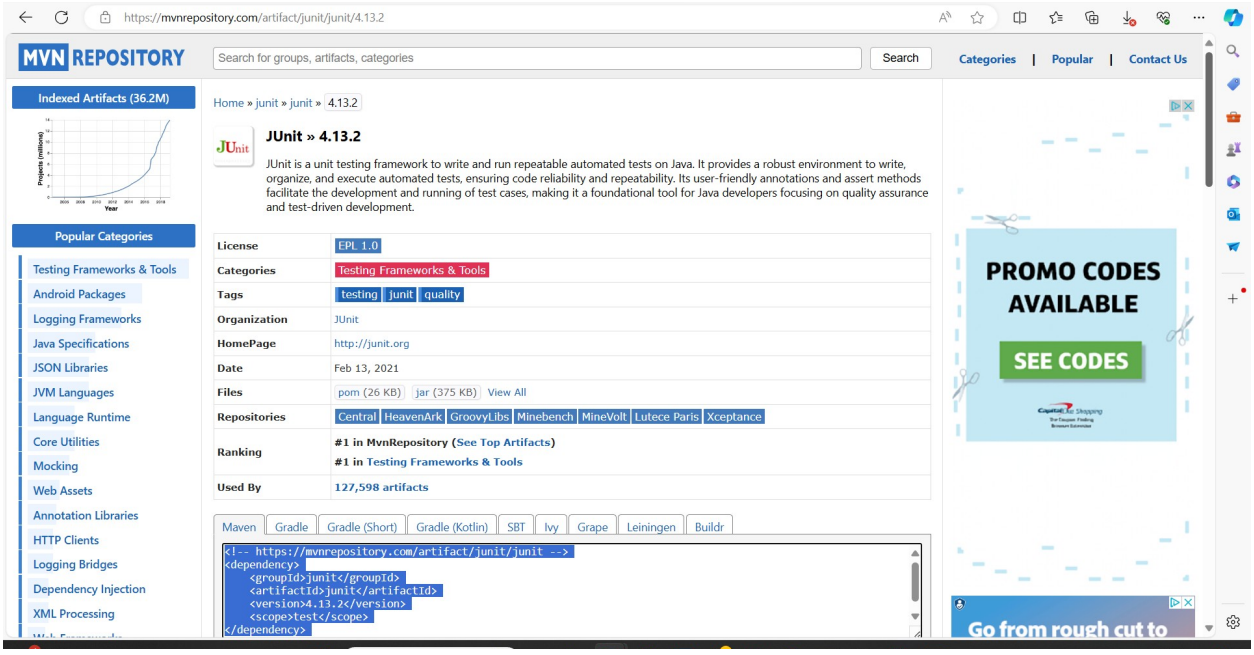
✅ Purpose of JUnit Dependency:

1. **Automate Testing:**
   - Run automated tests to check if methods, classes, and APIs work as expected.
2. **Assertions for Validation:**
   - Validate expected outcomes using assertions like:
     - `assertEquals()` – Checks if values are equal.
     - `assertTrue()` – Validates if a condition is `true`.
     - `assertNotNull()` – Ensures an object is not `null`.

3. **Test Execution Control:**
   - Control the execution flow using:
     - `@BeforeAll` – Runs once before all tests.
     - `@BeforeEach` – Runs before each test.
     - `@AfterEach` – Runs after each test.
     - `@AfterAll` – Runs once after all tests.
4. **Integration with REST Assured:**
   - JUnit is commonly used to **validate API responses** (e.g., **status codes, body content**) in REST Assured tests.

*** Status code:

```
response.prettyPrint();

System.out.println(response.getStatusCode());

Otherwise, we use :
//Get status code => it returns integer
                    int statusCode=response.getStatusCode();
                    System.out.println(statusCode);
```


https://mvnrepository.com/artifact/junit/junit/4.13.2
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (36.2M)
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Language Runtime
Core Utilities
Mocking
Web Assets
Annotation Libraries
HTTP Clients
Logging Bridges
Dependency Injection
XML Processing
Home » junit » junit » 4.13.2
JUnit » 4.13.2
JUnit is a unit testing framework to write and run repeatable automated tests on Java. It provides a robust environment to write, organize, and execute automated tests, ensuring code reliability and repeatability. Its user-friendly annotations and assert methods facilitate the development and running of test cases, making it a foundational tool for Java developers focusing on quality assurance and test-driven development.
License EPL 1.0
Categories Testing Frameworks & Tools
Tags testing junit quality
Organization JUnit
HomePage http://junit.org
Date Feb 13, 2021
Files pom (26 KB) jar (375 KB) View All
Repositories Central HeavenArk GroovyLibs Minebench MineVolt Lutece Paris Xceptance
Ranking #1 in MvnRepository (See Top Artifacts)
#1 in Testing Frameworks & Tools
Used By 127,598 artifacts
Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.13.2</version>
<scope>test</scope>
</dependency>
PROMO CODES AVAILABLE
SEE CODES
Go from rough cut to

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>4.13.2</version>
                  <scope>test</scope>
            </dependency>
```

You have to see on maven dependencies : JUNIT-4.132 JAR File

Do not forget to save the file !

Update the Project if you do not see the Jar files

You can add more dependencies:

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>API</groupId>

<artifactId>API</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

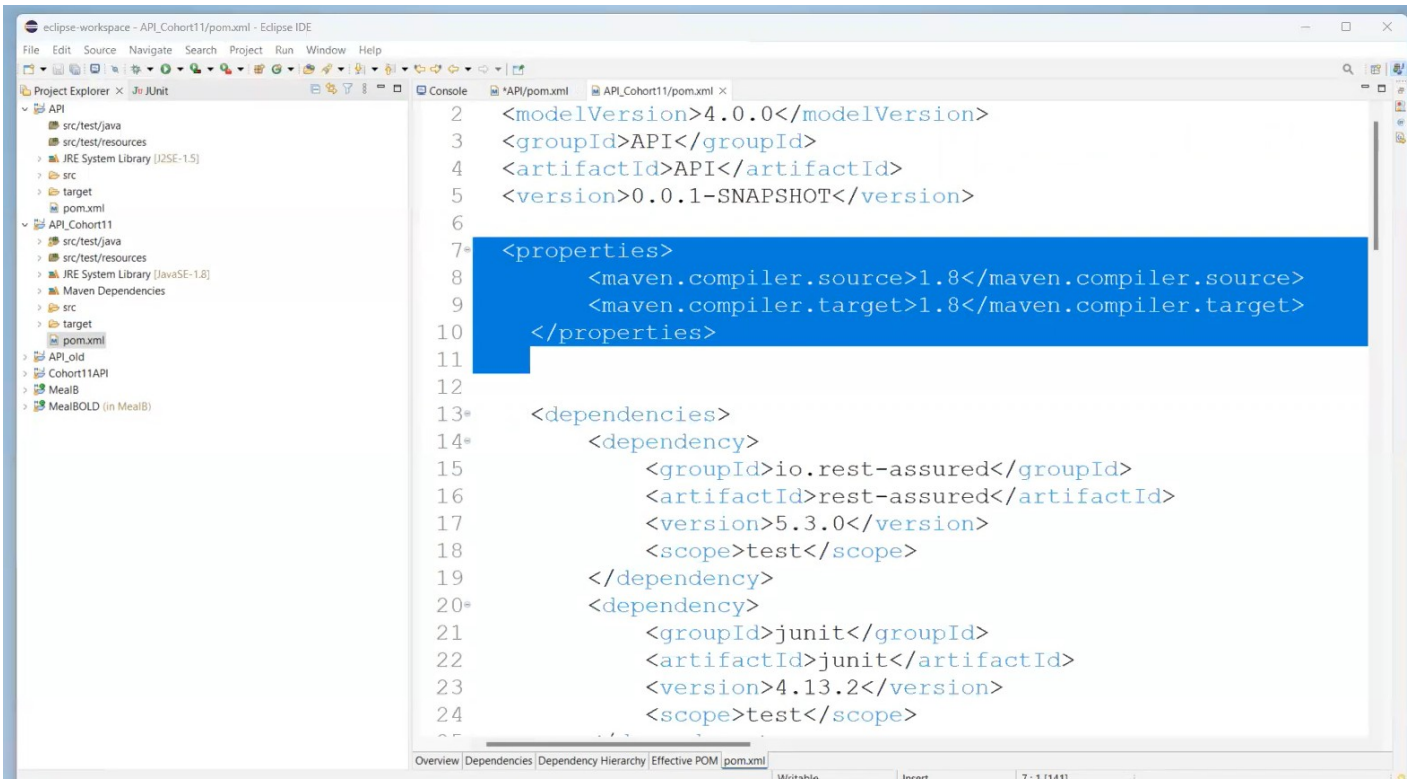
<dependencies>

<dependency>

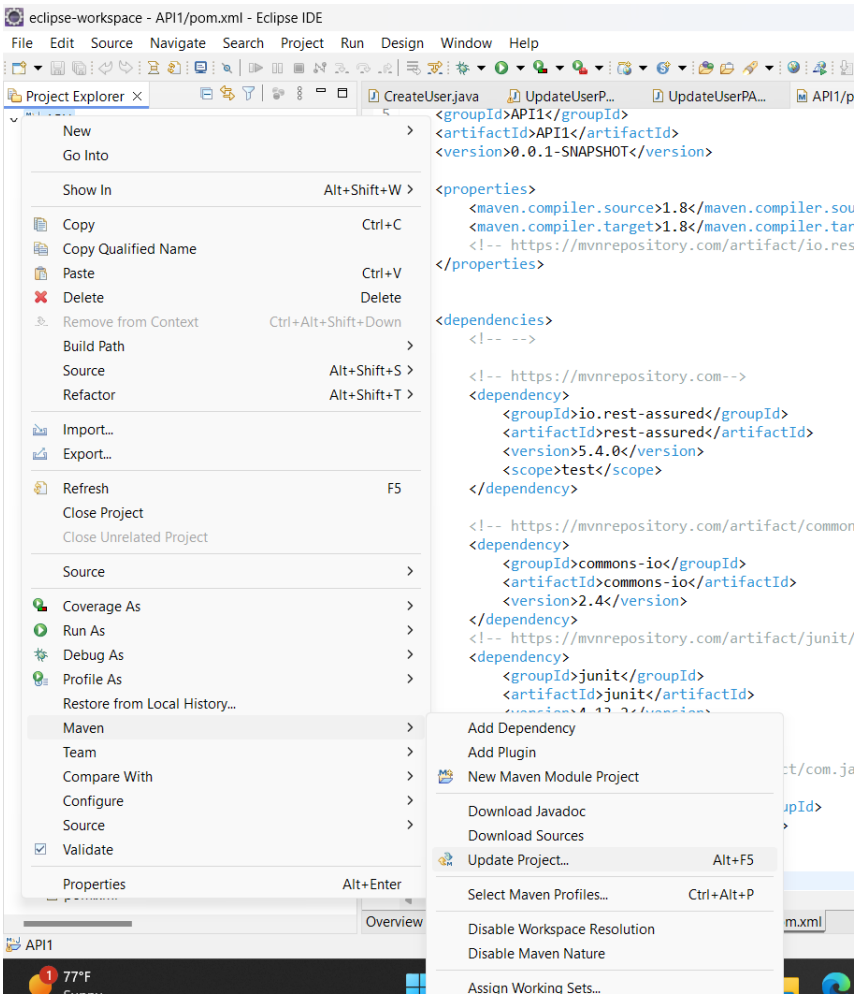
```
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>5.3.0</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.jayway.jsonpath</groupId>
        <artifactId>json-path</artifactId>
        <version>2.7.0</version>
    </dependency>
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.4</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.2</version>
    </dependency>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>json-schema-validator</artifactId>
        <version>5.3.0</version>
```

```
        </dependency>
    </dependencies>
</project>
```

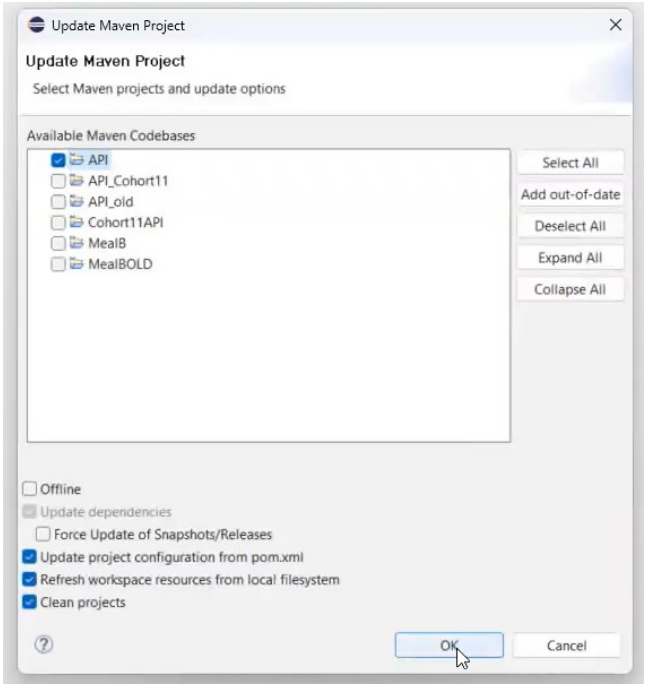
Save, you will see Maven Dependencies file on the left of eclipse



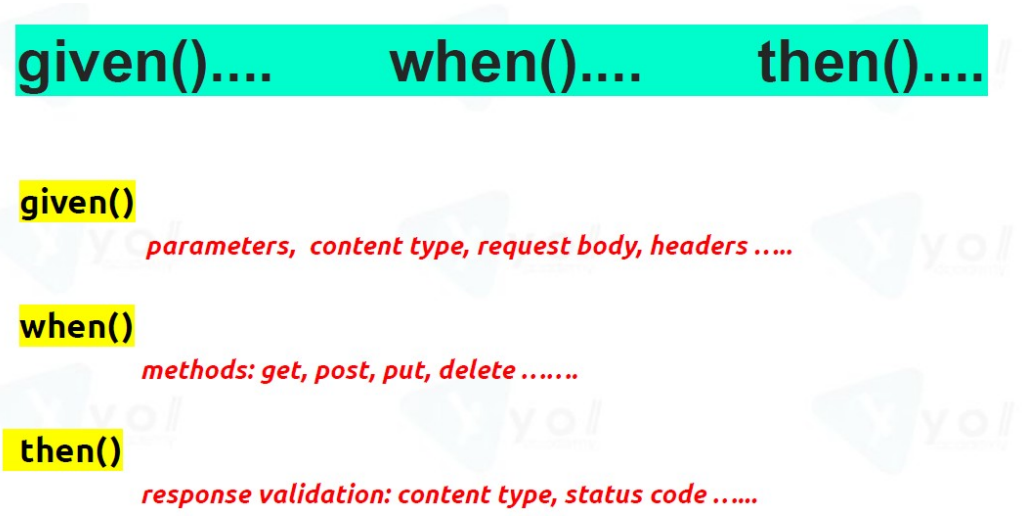
Update the Project: Right Click On the Java Project:



Then click on OK button:



The structure of a Rest Assured test in Java using the given() – when() – then() format.



The image illustrates the structure of a **Rest Assured** test in Java using the **given() – when() – then()** format. This structure aligns with the flow of making an API request and validating the response. Here's a breakdown of each part:

---

1. given()

- This section sets up the **preconditions** for the API call.
- Includes:
  - Parameters (query/path parameters)
  - Content-Type (JSON, XML, etc.)
  - Request body (for POST/PUT requests)
  - Headers (authentication tokens, etc.)

Example in Rest Assured:

```
given()
    .header("Content-Type", "application/json")
    .header("Authorization", "Bearer YOUR_ACCESS_TOKEN")
    .body("{ \"name\": \"John Doe\", \"email\": \"john@example.com\" }");
```

Example in Postman:

In **Postman**, you would:

1. Set the **Authorization** under the "Headers" tab (e.g., Bearer Token).
2. Add your **request body** under the "Body" tab (in raw JSON format).
3. Define query parameters under the "Params" tab.

---

2. when()

- This section defines the **HTTP method** (GET, POST, PUT, DELETE) and the **API endpoint**.
- It triggers the actual request.

Example in Rest Assured:

```
.when()
    .post("/users");
```

Example in Postman:

1. Choose **POST** (or any other method).
2. Provide the API endpoint (e.g., `https://gorest.co.in/public/v1/users`).
3. Click **Send** to execute the request.

---

3. then()

- This section is used to **validate** the response.
- Common assertions:
  - Status Code (200, 201, 400, etc.)
  - Content-Type (application/json)
  - Response body content

Example in Rest Assured:

```
.then()
    .statusCode(201)
    .body("name", equalTo("John Doe"))
    .body("email", equalTo("john@example.com"));
```

Example in Postman:

1. Use the **Tests** tab to write JavaScript-based assertions:

```
pm.response.to.have.status(201);
pm.response.to.have.jsonBody("name", "John Doe");
```

✅ Complete Example in Java (Rest Assured):

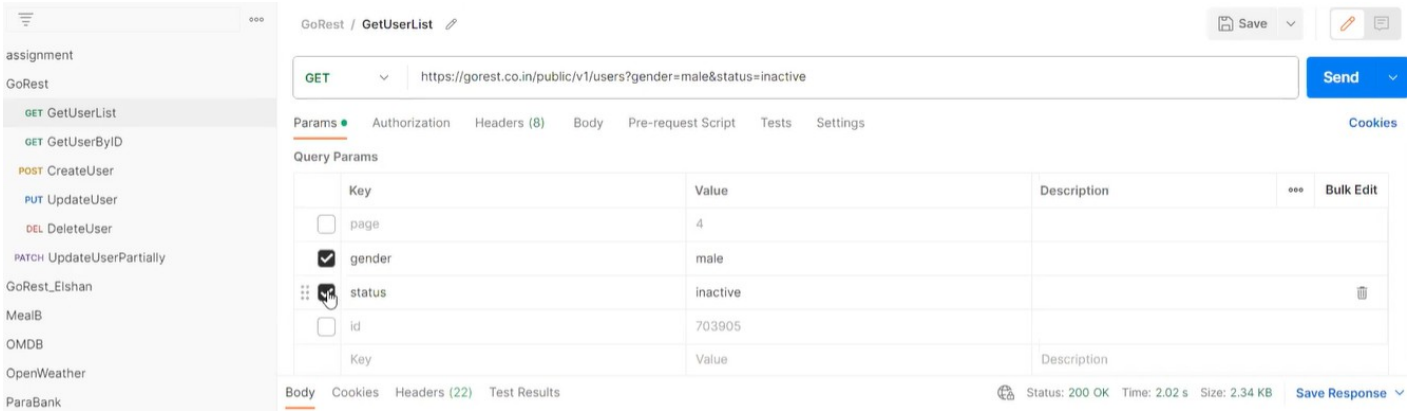
```
import io.restassured.RestAssured;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class CreateUserTest {
```

```
    public static void main(String[] args) {
        RestAssured.baseURI = "https://gorest.co.in/public/v1";

        given()
            .header("Content-Type", "application/json")
            .header("Authorization", "Bearer YOUR_ACCESS_TOKEN")
            .body("{ \"name\": \"John
Doe\", \"email\": \"john@example.com\", \"gender\": \"male\", \"status\": \"active\" }"
)
        .when()
            .post("/users")
        .then()
            .statusCode(201)
            .body("name", equalTo("John Doe"))
            .body("email", equalTo("john@example.com"));
    }
}
```

This script:

1. **given()** – Sets up headers and the request body.
2. **when()** – Sends a **POST** request to the `/users` endpoint.
3. **then()** – Validates the status code and response content.

Complete Example in Postman:



GET
REQUEST
given() method
given() >>> method is a static method in RestAssured class. So we can use it in two ways:
1. It can be called by class name:
Response response=RestAssured.given().when().get(url);
2. given() method can be imported in static way and used without class name
import static io.restassured.RestAssured.given;
Response response=given().when().get(url);

The `given()` method in **Rest Assured** is used to set up the **pre-conditions** for an API request. This method is part of the **RestAssured** class and is a **static method**. It can be used in two ways:

🔹 1. Using `given()` with Class Name

You can call the `given()` method by using the `RestAssured` class reference.

Example:

```
import io.restassured.RestAssured;
import io.restassured.response.Response;

public class GetUser {
    public static void main(String[] args) {
        Response response = RestAssured.given()
                .baseUri("https://gorest.co.in/public/v1")
                .when()
```

```
                .get("/users");

        System.out.println("Response: " + response.asString());
    }
}
```

✅ Explanation:

- `RestAssured.given()` – Accessing the `given()` method using the class name.
- `.when()` – Defines the action (GET, POST, PUT, DELETE).
- `.get("/users")` – Performs a GET request on the `/users` endpoint.

✅ 2. Using `given()` with Static Import

You can import the `given()` method statically to use it without the `RestAssured` class name.

Example:

```
import io.restassured.response.Response;
import static io.restassured.RestAssured.given;

public class GetUser {
    public static void main(String[] args) {
        Response response = given()
                .baseUri("https://gorest.co.in/public/v1")
                .when()
                .get("/users");

        System.out.println("Response: " + response.asString());
    }
}
```

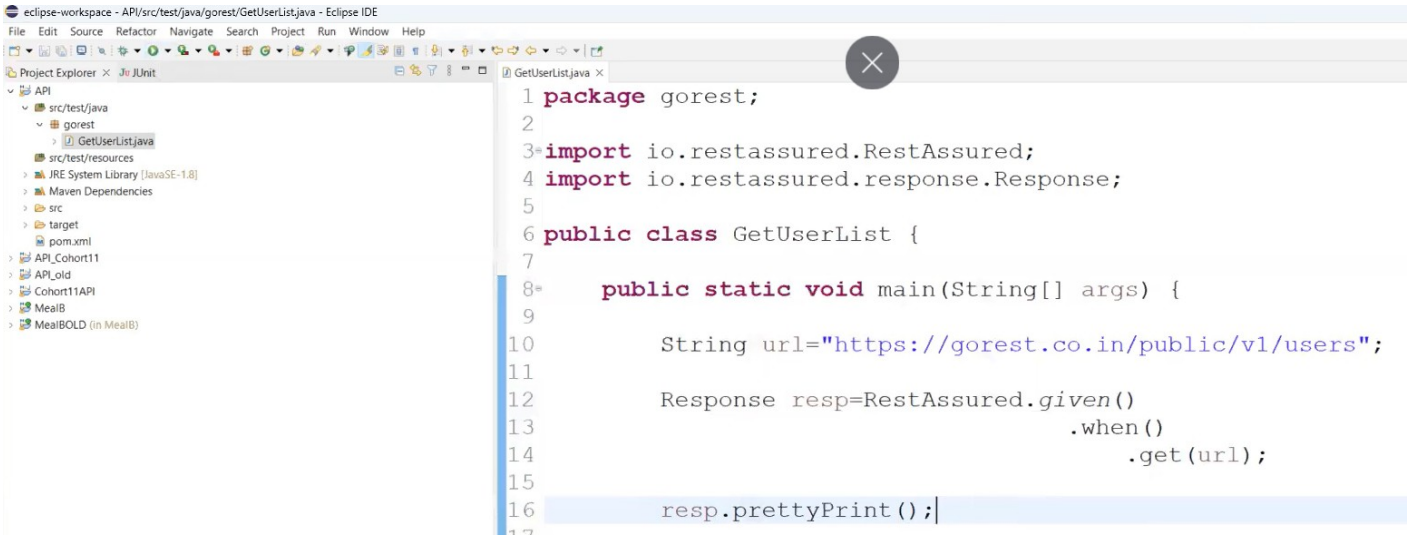
✅ Explanation:

- `import static io.restassured.RestAssured.given;` – Statically imports the `given()` method.
- We can directly call `given()` without prefixing it with `RestAssured`.
- This approach improves **code readability**.

✅ When to Use Which Approach?

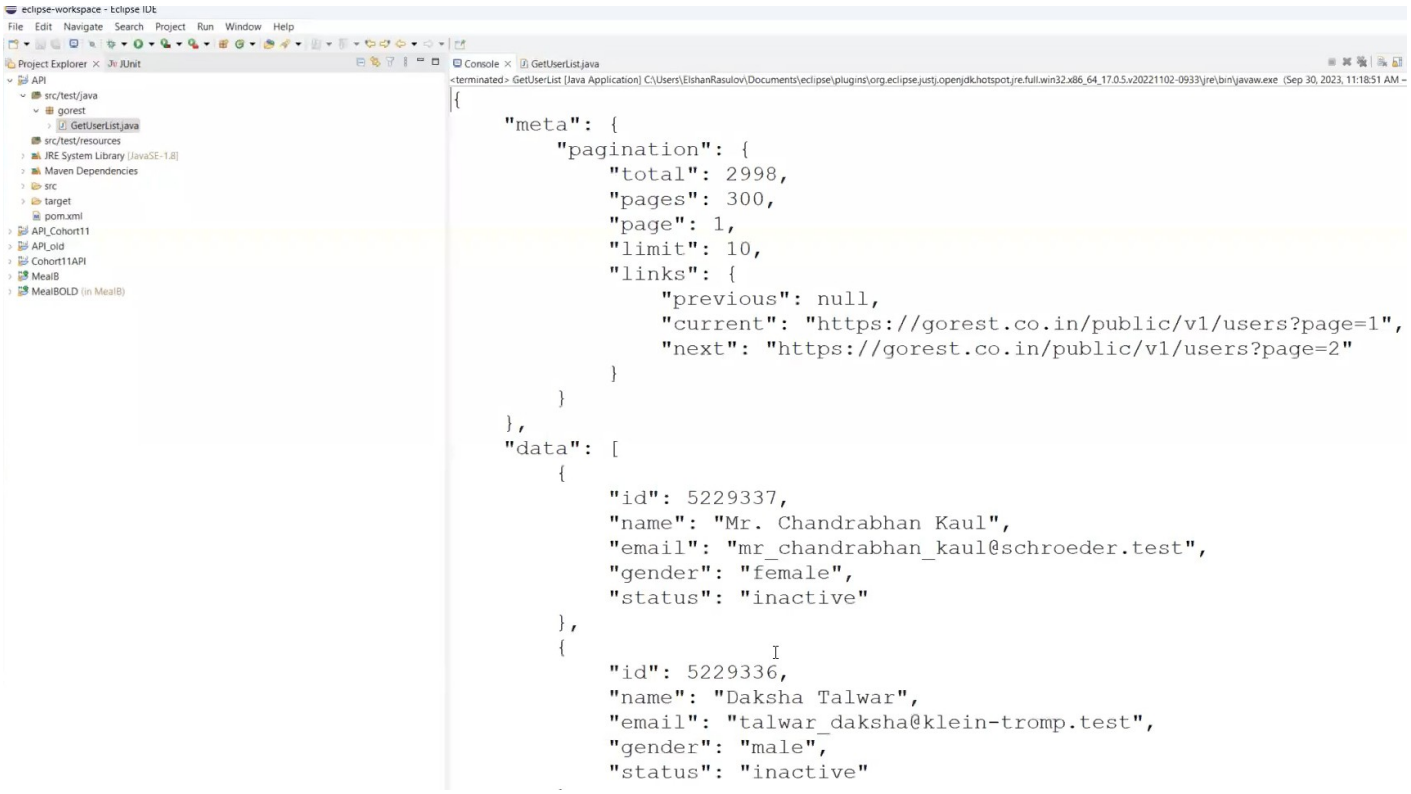
1. **With Class Name** (`RestAssured.given()`):
   - Use it when you want to **avoid additional imports**.
   - Ideal for **small test cases** or when **few tests** are written.
2. **With Static Import** (`given()`):
   - Use it when you have **multiple test cases**.
   - Improves code clarity and **reduces repetition**.

1- Example with the class Name:



```
package gorest;

import io.restassured.RestAssured;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

        String url="https://gorest.co.in/public/v1/users";

        Response resp=RestAssured.given()
                                 .when()
                                     .get(url);

        resp.prettyPrint();
```

Result:



```
{
    "meta": {
        "pagination": {
            "total": 2998,
            "pages": 300,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    },
    "data": [
        {
            "id": 5229337,
            "name": "Mr. Chandrabhan Kaul",
            "email": "mr_chandrabhan_kaul@schroeder.test",
            "gender": "female",
            "status": "inactive"
        },
        {
            "id": 5229336,
            "name": "Daksha Talwar",
            "email": "talwar_daksha@klein-tromp.test",
            "gender": "male",
            "status": "inactive"
```

```
package gorest;  // Package declaration

import io.restassured.RestAssured;          // Importing RestAssured class
import io.restassured.response.Response;    // Importing Response class

public class GetUserList {
```

```
    public static void main(String[] args) {
  // Base URL for the API
        String url = "https://gorest.co.in/public/v1/users";

// Sending GET request and storing the response
        Response resp = RestAssured.given()
                                .when()
                                .get(url);

// Printing the response in a formatted way
        resp.prettyPrint();
    }
                                                    }
```

🔍 Explanation of Each Section:

---

1. `package gorest;`

- **Purpose:** Defines the package name as `gorest`.
- **Why use it?** Helps organize Java classes into groups and avoid naming conflicts.

---

2. `import` statements

```
import io.restassured.RestAssured;
import io.restassured.response.Response;
```

- `RestAssured`: Main class from the **Rest Assured** library to make HTTP requests.
- `Response`: Captures the API response (status code, body, headers, etc.).

---

3. `public class GetUserList`

- **Defines the class** named `GetUserList`.
- This is the main Java class where you perform API operations.

---

4. `public static void main(String[] args)`

- **Entry point** of the Java program.
- Executes the code inside this method when the program runs.

---

5. `String url = "https://gorest.co.in/public/v1/users";`

- Stores the **API endpoint URL** in a variable `url`.

---

6. `Response resp = RestAssured.given().when().get(url);`

- **`RestAssured.given()`**: Prepares the API request.
- **`.when()`**: Defines the action (GET, POST, PUT, DELETE).
- **`.get(url)`**: Sends a **GET** request to the specified URL.
- **`Response resp`**: Stores the API response.

---

7. `resp.prettyPrint();`

- Prints the **response body** in a **formatted (readable)** manner.

✅ Output Example:

When you run this code, it might print a JSON response like:


```
{
    "meta": {
        "pagination": {
            "total": 3000,
            "pages": 300,
            "page": 1,
            "limit": 10
        }
    },
    "data": [
        {
            "id": 123,
            "name": "John Doe",
            "email": "john.doe@example.com",
            "gender": "male",
            "status": "active"
        },
        {
            "id": 124,
            "name": "Jane Doe",
            "email": "jane.doe@example.com",
            "gender": "female",
            "status": "inactive"
        }
    ]
}
```

✅ How to Add Assertions (Optional Validation)

You can validate the response using `.then()` and add assertions:

```
Response resp = RestAssured.given()
    .when()
    .get(url)
    .then()
    .statusCode(200)   // Assert that status code is 200 (OK)
    .extract().response();

resp.prettyPrint();
```

✅ Make Sure You Have the Rest Assured Dependency:

If you're using **Maven**, add this to your `pom.xml`:

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.4.0</version>
    <scope>test</scope>
</dependency>
```

This ensures that **Rest Assured** is available in your project.

2- Example with Static Import:

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given().when().get(url);

          // print response data
          response.prettyPrint();


    }

}
```

package gorest; // Defines the package name

import static io.restassured.RestAssured.given; // Static import for 'given()' method

import io.restassured.response.Response; // Import Response class to handle API responses

public class GetUserList {

public static void main(String[] args) {

// Define the API endpoint URL

String url = "https://gorest.co.in/public/v1/users";

// Submit GET request and receive response

Response response = given()

.when()

.get(url);

// Print the response body in a readable format

```
        response.prettyPrint();

    }
}
```

2. `import static io.restassured.RestAssured.given;`

- **Static Import** allows us to directly call the `given()` method without needing to prefix it with `RestAssured`.

Without static import:

```
Response response = RestAssured.given().when().get(url);
```

With static import:

```
Response response = given().when().get(url);
```

✅ Why use static import?

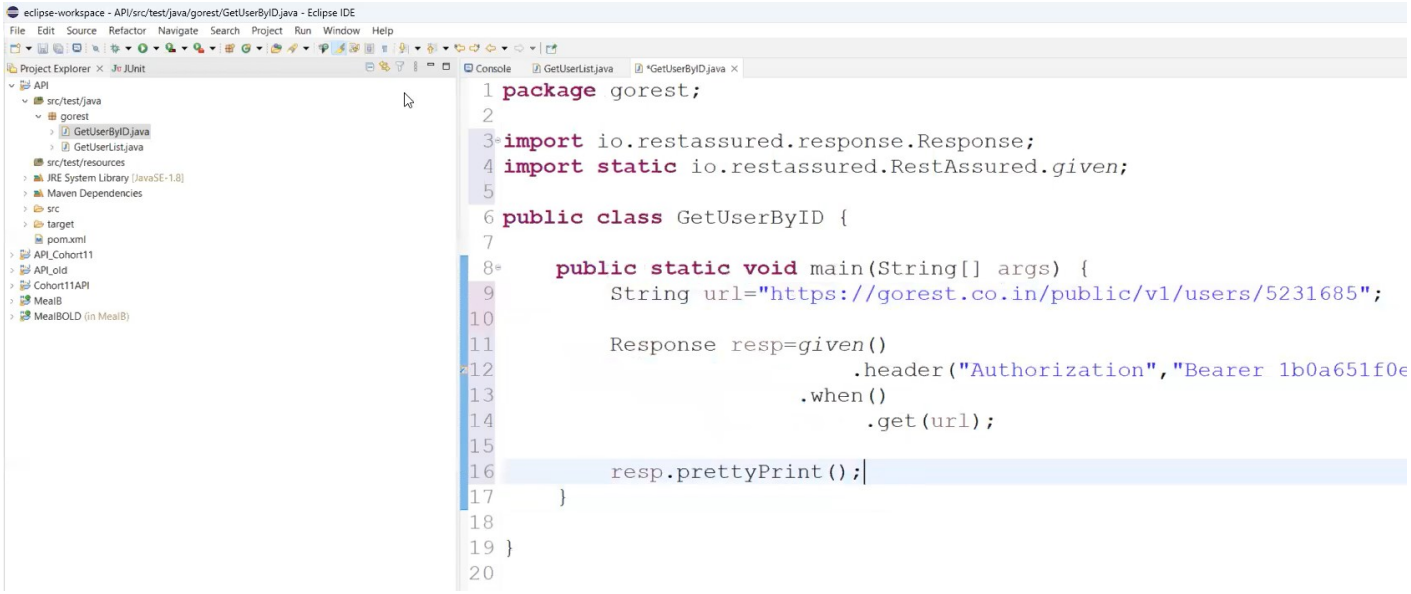
- Makes the code cleaner and easier to read.
- Avoids repeatedly writing the class name (`RestAssured`).

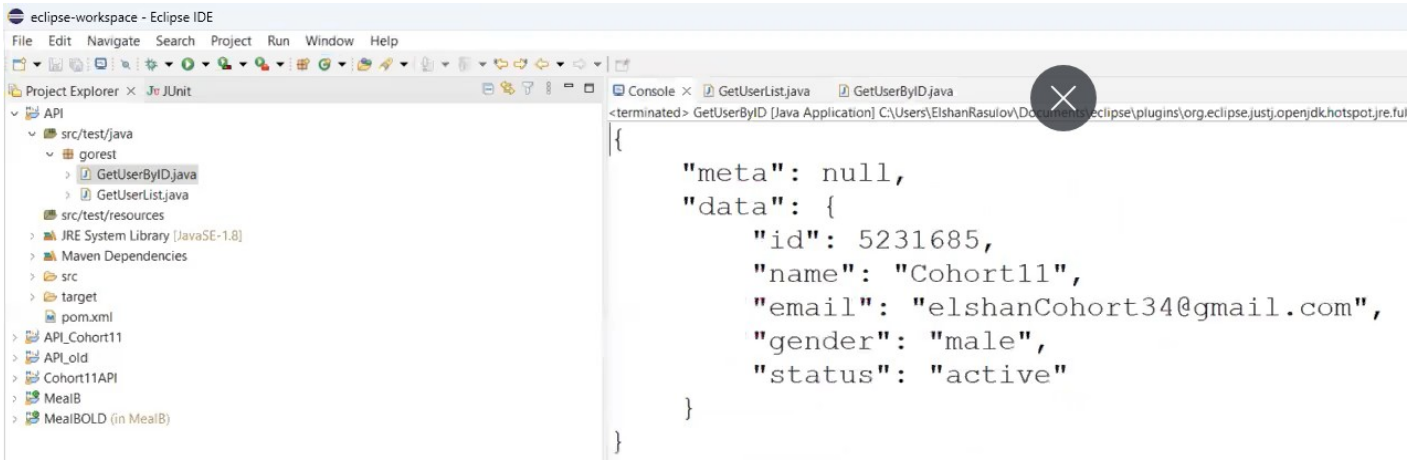
---

3. `import io.restassured.response.Response;`

- **Response** is a class in Rest Assured used to capture the **HTTP response** from the API.
- This object contains:
  - **Status Code** (e.g., 200, 404, 500)
  - **Response Body** (JSON, XML, HTML, etc.)
  - **Headers**
  - **Cookies**

Example of getting the user :5231685





Path and Query Parameters in REST Assured

QUERY PARAMETERS: are passed in KEY + VALUE format.
In automation we use .param(key, value) method to pass query parameters. Query parameters are separated from URL by question mark (?).

PATH PARAMETER: is a part of url and added to the end of the url. Usually record id is used as path parameters to retrieve a specific record from database with GET request:
Note: Record ID is a unique ID for each record added to database. It is generated by system

When working with **APIs**, both **query** and **path** parameters are used to pass information to the server, but they serve different purposes:

Parameter Type	Description	Example
Path Parameter	Identifies a specific resource in the URL.	`/users/{id}` (e.g., `/users/123`)
Query Parameter	Filters or customizes the request's response.	`/users?gender=male&status=active`

ONE PARAMETER:

.param(): Adds query parameters to the request.

This filters the API response:

- **"gender", "female" → Filters for female users.**
- **"status", "active" → Filters for active users.**

Get female users:

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given()
                                .param("gender", "female")
                            .when()
                                .get(url);

          // print response data
          response.prettyPrint();


    }

}
```

```
package gorest;
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {
    public static void main(String[] args) {
         // API Endpoint
        String url = "https://gorest.co.in/public/v1/users";
        // Submit GET request with query parameters and receive response
        Response response = given()
                                .param("gender", "female")      // Add one query parameter
                        .when()
                            .get(url);                  // Send GET request

        // Print response data
        response.prettyPrint();
    }
}
```

Result :

Two PARAMETERS:

```
package gorest;
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {
    public static void main(String[] args) {
        // API Endpoint
        String url = "https://gorest.co.in/public/v1/users";
        // Submit GET request with query parameters and receive response
        Response response = given()
                            .param("gender", "female")      // Add query parameter
                            .param("status", "active")      // Add another query
parameter
                        .when()
                            .get(url);                  // Send GET request

        // Print response data
        response.prettyPrint();
```

```
    }
}
```

✅ Alternative Method Using `.queryParam()`

You can also use `.queryParam()` instead of `.param()` for better clarity.

```
Response response = given()
                        .queryParam("gender", "female")
                        .queryParam("status", "active")
                    .when()
                        .get(url);
```

Both `.param()` and `.queryParam()` work the same way for **GET** requests.

header() method is used pass headers to request.
username, password, Token, ContentType ... can be passed as header.

```
String url="https://gorest.co.in/public/v1/users";

Response response=given()
                        .param("gender", "female")
                        .header("Authorization","Bearer a4kdsu4981")
                    .when()
                        .get(url);
```

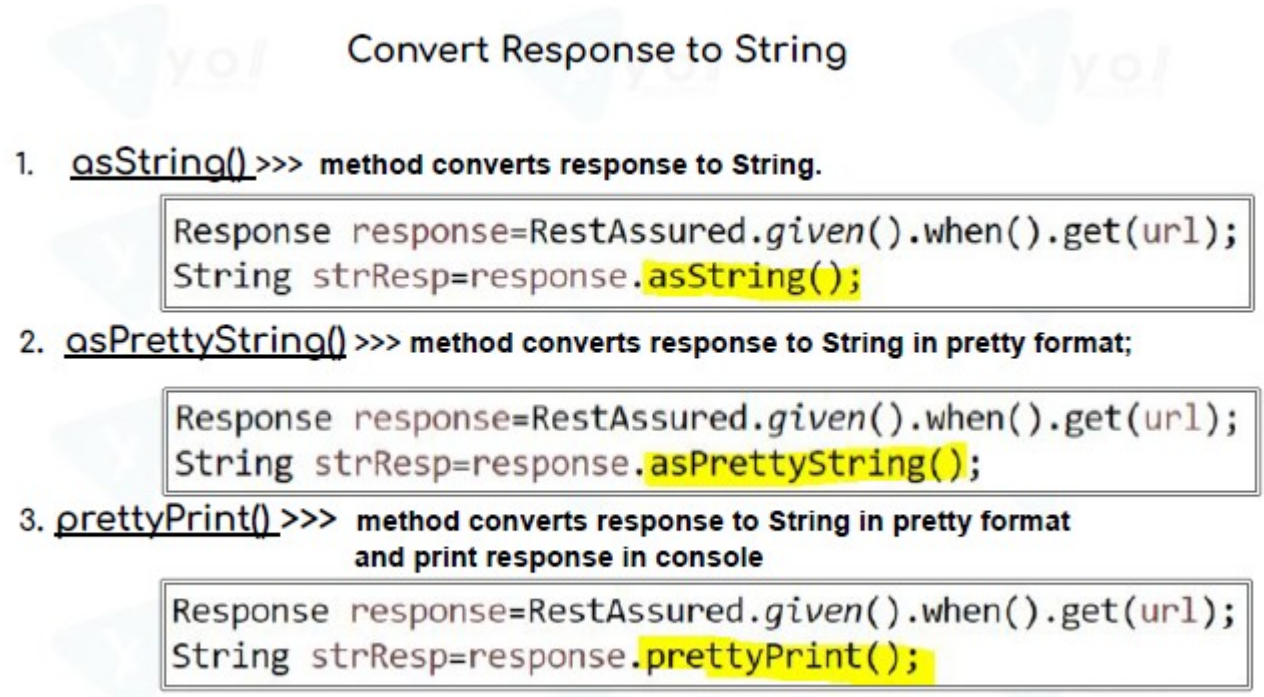
Adding Authorization:

```
public static void main(String[] args) {

      String url="https://gorest.co.in/public/v1/users";
     // submit API GET request and receive response
      Response response= given()
                            .param("gender", "female")
                            .param("status", "active")
                            .header("Authorization","Bearer 1b0a651f0ee6151ed5586b675cd871b4178f23f1ee6900f0fb7a34d787ac1855")
                        .when()
                            .get(url);
```

To **add authorization** (like a **Bearer Token**) in your Rest Assured request, you can include it using the .header() method or .auth() method.

Submit GET request with query parameters and Authorization

Response response = given()
.header("Authorization", "Bearer + token") **// Add Bearer Token**
.**queryParam**("gender", "female") **// Query Parameter .queryParam("status", "active")**


Convert Response to String
1. asString() >>> method converts response to String.
Response response=RestAssured.given().when().get(url);
String strResp=response.asString();
2. asPrettyString() >>> method converts response to String in pretty format;
Response response=RestAssured.given().when().get(url);
String strResp=response.asPrettyString();
3. prettyPrint() >>> method converts response to String in pretty format and print response in console
Response response=RestAssured.given().when().get(url);
String strResp=response.prettyPrint();

The image explains three different methods to convert the API response to a **String** format using **Rest Assured** in Java:

---

1. `asString()`

- **Purpose**: Converts the **raw response** to a plain **String**, and **needs `System.out.println()`**

 Example:

```
Response response = RestAssured.given().when().get(url);
String strResp = response.asString();
System.out.println(strResp);
```

 Output (Unformatted JSON/String):

```
{"id":123,"name":"John Doe","gender":"male"}
```

---

2. `asPrettyString()`

- **Purpose**: Converts the response to a **formatted (pretty)** JSON/XML **String**, , and **needs `System.out.println()`**

 Example:

```
Response response = RestAssured.given().when().get(url);
String strResp = response.asPrettyString();
System.out.println(strResp);
```

 Output (Well-formatted JSON/XML):

```
{
  "id": 123,
  "name": "John Doe",
  "gender": "male"
}
```

---

3. `prettyPrint()` It is the best for printing

- **Purpose**: Converts the response to a **pretty-formatted** String and **prints** it directly to the console.

 Example:

```
Response response = RestAssured.given().when().get(url);
response.prettyPrint();  // Directly prints to console and it does need System.out.println() like : asPrettyString() and asString()
```

 Output (Pretty-printed JSON/XML to console):

```
{
  "id": 123,
  "name": "John Doe",
  "gender": "male"
}
```

 Key Differences:

Method	Output Type	Pretty Formatted	Automatically Prints to Console
`asString()`	Plain String	No	No
`asPrettyString()`	Formatted String	Yes	No

Method	Output Type	Pretty Formatted	Automatically Prints to Console
`prettyPrint()`	Formatted String	✅ Yes	✅ Yes

✅ When to Use Each Method:

- **`asString()`**: If you need to **process** the raw response (e.g., for assertions or saving).
- **`asPrettyString()`**: If you need to **process** the formatted response (for better human readability).
- **`prettyPrint()`**: If you only want to **log** the response for **debugging**.

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given()
                              .param("gender", "female")
                              .param("status", "active")
                          .when()
                              .get(url);

          // print response data and convert it to String
          //String strResponse= response.prettyPrint();

          //convert response to String
        // String strResponse2= response.asString();
        // System.out.println(strResponse2);

          //convert response to String in pretty format
          String strResponse3= response.asPrettyString();
         System.out.println(strResponse3);
    }

}
```

Get the user: 7733983

```
package gorest;

import io.restassured.response.Response;
import static io.restassured.RestAssured.given;
```

```
import io.restassured.http.ContentType;
public class GetUserByID {
	public static void main(String[] args) {
		String url = "https://gorest.co.in/public/v1/users/7733983";
		Response resp=given()
					.header("Authorization","Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
					//Using User Name and Password for Basic Auth on
Postman
					//.header("Username","john")
					//.header("Password","demo")

					.when()
					.get(url);
		resp.prettyPrint();
		//Validate Assertion method
		//resp.then().statusCode(200);
		//Get status code => it returns integer
		int statusCode=resp.getStatusCode();
		System.out.println(statusCode);
		// validate response data/content type.
		//To make sure that the content is on JSON not on another format
		//First Way
		resp.then().contentType("application/json");
		//Second Way
		resp.then().contentType(ContentType.JSON);
		//Extract value from response
		String email=resp.then().extract().path("data.email");
		System.out.println(email);
		int id=resp.then().extract().path("data.id");
		System.out.println(id);



	}
}
```

Get the list of the users

```
package gorest;

import io.restassured.RestAssured;
import io.restassured.response.Response;

public class GetUserList {
public static void main(String[] args) {
	String url="https://gorest.co.in/public/v1/users";

	Response resp=RestAssured.given()
						.param("gender", "male")
						.param("status", "inactive")

			.when().get(url);
	resp.prettyPrint();

	//Convert resp to pretty String
	String q =resp.asPrettyString();
	System.out.println(q);
	//Convert resp to string and Print
	String strResponse =resp.prettyPrint();
}
```

```
}

/*Another way without using static


  package gorest;

import io.restassured.response.Response;
import static io.restassured.RestAssured.given;

public class GetUserList {
public static void main(String[] args) {
      String url="https://gorest.co.in/public/v1/users";

      Response resp=given()
                  .when()
                        .get(url);
      resp.prettyPrint();
}
}

  */
```

```
package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

        // API endpoint URL

        String url = "https://gorest.co.in/public/v1/users";

        // Submit GET request with query parameters and receive response

        Response response = given()

                .param("gender", "female")   // Add query parameter: gender=female

                .param("status", "active")   // Add query parameter: status=active

                .when()

                .get(url);
```

```
        // 1. Print response data in pretty format to console
        response.prettyPrint();

        // 2. Convert response to raw String and print
        String strResponse = response.asString();
        System.out.println("Raw Response: \n" + strResponse);

        // 3. Convert response to pretty String and print
        String strResponsePretty = response.asPrettyString();
        System.out.println("Pretty Response: \n" + strResponsePretty);
    }
}
```

when()

After **when()** we write one of request methods: ***get()***, **post()**, **put()**, **delete()**

```
Response response=given()
                        .param("gender", "female")
                        .header("Authorization","Bearer a4kdsu4981")
                    .when()
                         .get(url);
```

In **Rest Assured**, the `.when()` method is used to specify the **HTTP request method** you want to execute. It follows the setup of request parameters, headers, or authentication and precedes the actual request (e.g., `GET()`, `POST()`, `PUT()`, `DELETE()`).

GET

when() // Triggering the HTTP method

.get(url);

POST

.when()

.post(url);

PUT

.when()

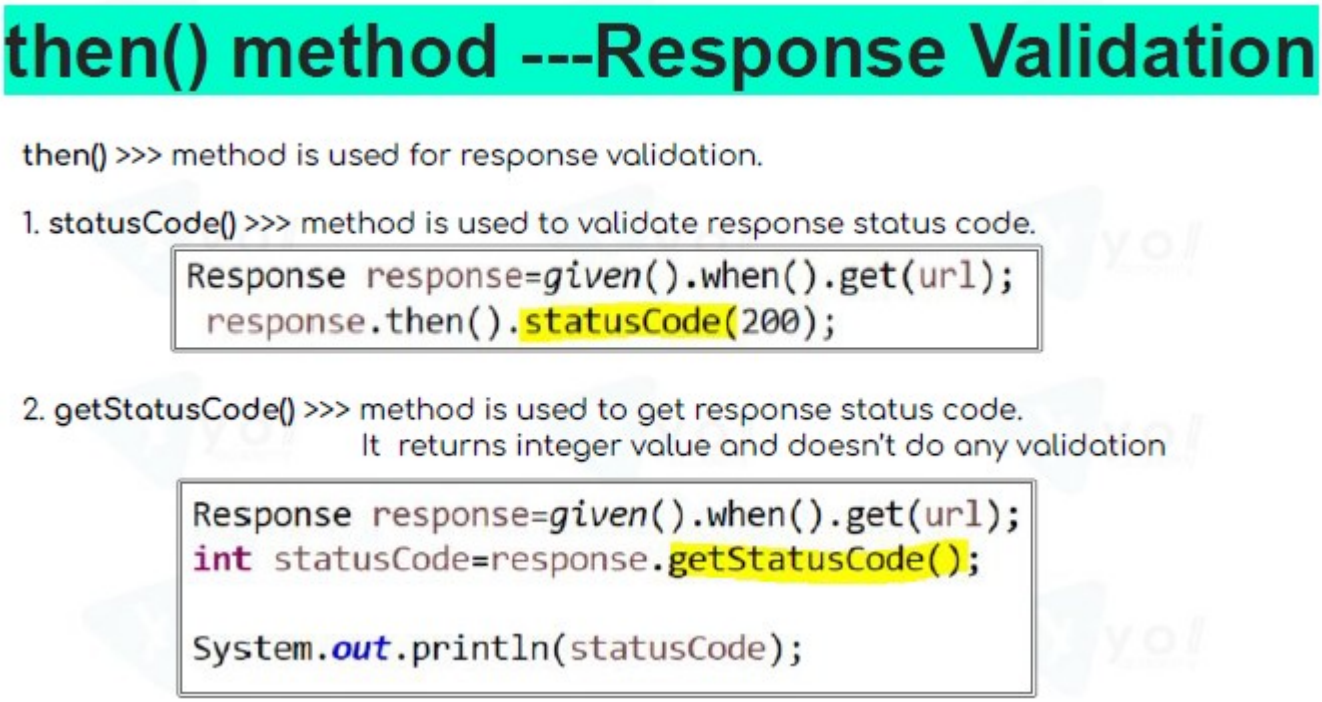
.put(url);

DELETE

when()

.delete(url);

then() method in Rest Assured


then() method ---Response Validation
then() >>> method is used for response validation.
1. statusCode() >>> method is used to validate response status code.
Response response=given().when().get(url);
response.then().statusCode(200);
2. getStatusCode() >>> method is used to get response status code.
It returns integer value and doesn't do any validation
Response response=given().when().get(url);
int statusCode=response.getStatusCode();
System.out.println(statusCode);

The `then()` method in **Rest Assured** is primarily used to **validate and assert** the API response. It allows you to check whether the API response meets the expected conditions.

🔹 1. `statusCode()` Method

You need to use THEN()

- **Purpose**: **Validates** that the **HTTP status code** returned by the API matches the expected value.

Example:

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class ValidateStatusCode {
    public static void main(String[] args) {
        String url = "https://gorest.co.in/public/v1/users";
```

```
        // Submit GET request and validate status code
        Response response = given()
                .when()
                .get(url);

        // Validate status code is 200 (OK)
        response.then().statusCode(200);

        System.out.println("Status code is validated
successfully!");
    }
}
```

Explanation:

1. `.when().get(url)` – Sends a **GET** request to the specified URL.
2. `.then()` – Begins the validation section.
3. `.statusCode(200)` – Asserts that the returned HTTP status code is **200** (OK). If it's not, the test will **fail**.

Common Status Codes:

- **200** – OK (Successful response)
- **201** – Created (Resource successfully created)
- **400** – Bad Request (Invalid request syntax)
- **401** – Unauthorized (Missing/Invalid authorization)
- **404** – Not Found (Resource not found)
- **500** – Internal Server Error (Server issue)

---

2. `getStatusCode()` Method

You Do not need to use THEN()

- **Purpose**: **Retrieves the HTTP status code from the response as an integer, without performing any validation.**

Example:

```
package gorest;
```

```java
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class FetchStatusCode {
    public static void main(String[] args) {
        String url = "https://gorest.co.in/public/v1/users";

        // Submit GET request and get response
        Response response = given()
                .when()
                .get(url);

        // Fetch and print the status code
        int statusCode = response.getStatusCode();
        System.out.println("Status Code: " + statusCode);
    }
}
```

Explanation:

1. `.getStatusCode()` – Fetches the actual HTTP status code as an integer.
2. **No validation** is performed, so it won't fail even if the status code is unexpected.

✅ Difference Between `statusCode()` and `getStatusCode()`:

Method	Purpose	Validation Performed?	Return Type
`.statusCode()`	**Validates** expected status code	✅ **Yes** (Throws error if mismatch)	N/A (Assertion)
`.getStatusCode()`	**Fetches** the actual status code	❌ **No** (Only retrieves value)	`int` (Integer)

Example with Status Validation :

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given()
                                .param("gender", "female")
                                .param("status", "active")
                            .when()
                                .get(url);



          //response.then().statusCode(200);


          // print response data and convert it to String
          String strResponse= response.prettyPrint();
```

statusCode(200) Method:

The following code passes because we are expecting statusCode : 200

package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class ValidateAuth {

public static void main(String[] args) {

String url = "https://gorest.co.in/public/v1/users";

Response response = given()

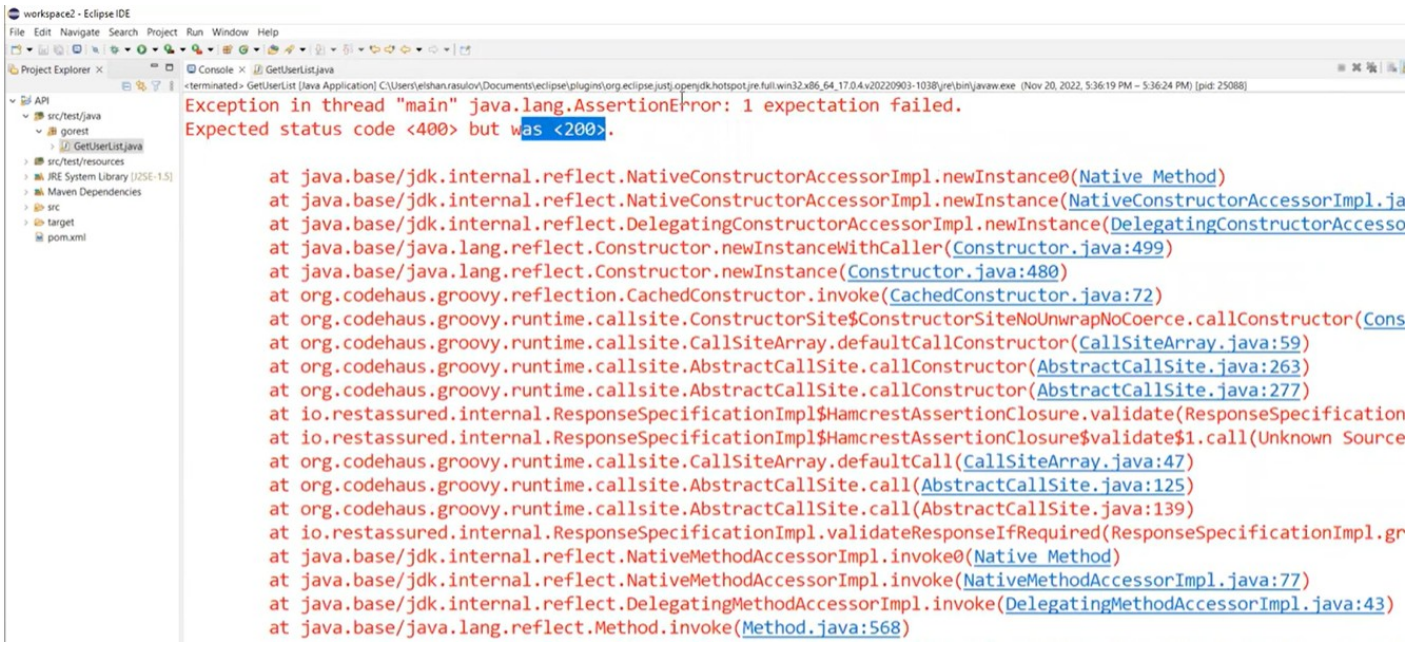
```
            .header("Authorization", "Bearer
YOUR_ACCESS_TOKEN")
            .when()
            .get(url);
            // Validate the status code
        response.then().statusCode(200);
            // Print actual status code
        System.out.println("Actual Status Code: " +
response.getStatusCode());
    }
}
```

Example with statusCode(400) Method:

The following code **fails** because we are expecting statusCode : **400**, **but the actual** statusCode is **200**

```
package gorest;
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class ValidateAuth {
    public static void main(String[] args) {
        String url = "https://gorest.co.in/public/v1/users";
            Response response = given()
            .header("Authorization", "Bearer
YOUR_ACCESS_TOKEN")
            .when()
            .get(url);
```

```
        // Validate the status code
        response.then().statusCode(400);
        // Print actual status code
        System.out.println("Actual Status Code: " +
response.getStatusCode());
    }
}
```



Example getStatusCode() Method

```
package gorest;
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;
public class FetchStatusCode {
    public static void main(String[] args) {
        String url = "https://gorest.co.in/public/v1/users";
```

```
        // Submit GET request and get response
        Response response = given()
                .when()
                .get(url);


         // Fetch and print the status code
        int statusCode = response.getStatusCode();
        System.out.println("Status Code: " + statusCode);
    }
}
```

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

        String url="https://gorest.co.in/public/v1/users";
        // submit API GET request and receive response
        Response response= given()
                            .param("gender", "female")
                            .param("status", "active")
                        .when()
                            .get(url);

        int actualStatusCode=response.getStatusCode();
        System.out.println(actualStatusCode);

        //response.then().statusCode(200);


        // print response data and convert it to String
        String strResponse= response.prettyPrint();
```

statusCode just prints the status code :200 without validating :

contentType() Method in Rest Assured – Response Validation

3. contentType() >>> when **contentType()** method is used after **then()** method, it validates format of response data. Data format can be passed in two ways into **contentType()** method for validation

1. **ContentType.JSON**
2. **"application/json"**

```
Response response=given().when().get(url);

response.then().contentType(ContentType.JSON);
response.then().contentType("application/json");
```

contentType() method is used to show format of request data.

```
Response response=given().param("q", "London")
                         .param("appid", "673c5650a2031104")
                         .contentType(ContentType.JSON)
                    .when().get();
```

Note: Content Type can also be passed as a **header**() method.

```
Response response=given().param("q", "London")
                         .param("appid", "673c5650a2031104")
                         .header("Content-Type", "application/json")
                    .when().get();
```

Example of paasing the Content-Type as a header:

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.response.Response;

public class CreateUser {

    public static void main(String[] args) {
        // API endpoint for creating a user
        String url = "https://gorest.co.in/public/v1/users";

        // JSON payload for the request
        String requestData = "{"
                + "\"name\": \"APIYo11Academy\","
                + "\"email\": \"yo11academy123@yahoo.com\","
                + "\"gender\": \"male\","
                + "\"status\": \"active\""
                + "}";

        // Send POST request with headers and JSON payload
        Response response = given()
                .header("Authorization", "Bearer YOUR_ACCESS_TOKEN_HERE") //
Replace with your actual token
                .header("Content-Type", "application/json") //
Content-Type in header
                .body(requestData)
                .post(url);

        // Print the response status code and body
        System.out.println("Status Code: " + response.getStatusCode());
        System.out.println("Response Body: " + response.getBody().asString());
```

```
    }
}
```

The `contentType()` method in **Rest Assured** is used to **validate** the **Content-Type** of the API response. This ensures the response data format matches the expected type.

✅ How to Use `contentType()` Method

1. **With ContentType**
   - Use :

     `ContentType.JSON`,

     `ContentType.XML`,

     `ContentType.TEXT`, etc.,

     from the **`io.restassured.http.ContentType`** package.

2. **With MIME Type String**
   - Use a **String** representation of the content type **(e.g.,**
   - **`"application/json",`**
   - **`"application/xml",`**
   - **`"text/plain").`**

✅ Common Content Types:

Content Type	Enum Equivalent	Purpose
`application/json`	`ContentType.JSON`	JSON response
`application/xml`	`ContentType.XML`	XML response
`text/plain`	`ContentType.TEXT`	Plain text response
`application/pdf`	N/A	PDF document
`application/octet-stream`	N/A	Binary data (file download)

✅ Example with Authorization and Content-Type Validation

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class ValidateAuthContentType {
    public static void main(String[] args) {
        String url = "https://gorest.co.in/public/v1/users";

        Response response = given()
                .header("Authorization", "Bearer
YOUR_ACCESS_TOKEN")
                .when()
                .get(url);

        // Validate JSON content type
        response.then().contentType(ContentType.JSON);

        System.out.println("Response content type is validated
successfully!");
    }
}
```

✅ When to Use `contentType()`:

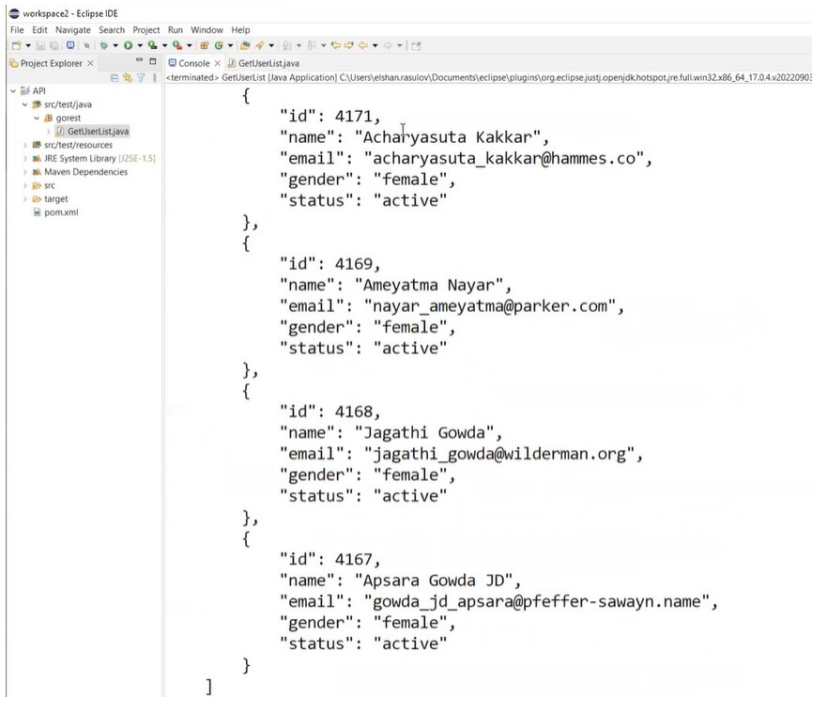
1. **Validate API responses** to ensure they return the expected format (JSON, XML, etc.).
2. **Prevent errors** when parsing responses by ensuring proper content-type handling.
3. **Automated testing** where content validation is required in API test cases

Task :

Validate the expected data and the actual data with JSON FORMAT

```java
package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

        String url="https://gorest.co.in/public/v1/users";
        // submit API GET request and receive response
        Response response= given()
                            .param("gender", "female")
                            .param("status", "active")
                        .when()
                            .get(url);

        response.then().contentType(ContentType.JSON);


//      int actualStatusCode=response.getStatusCode();
//      System.out.println(actualStatusCode);

        //response.then().statusCode(200);


        // print response data and convert it to String
        String strResponse= response.prettyPrint();
```

When we run it, it passes, which means: the format is validated by using ContentType.Json because the actual format is the expected format



Negative Scenario:

Expected data :with XML Format

Actual Data :with JSON Format

```
package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given()
                                .param("gender", "female")
                                .param("status", "active")
                            .when()
                                .get(url);

          response.then().contentType(ContentType.xml);


//        int actualStatusCode=response.getStatusC
//        System.out.println(actualStatusCode);

          //response.then().statusCode(200);


          // print response data and convert it to
          String strResponse= response.prettyPrint();
```

Result:

We got an error, the Content format is supposed to be JSON Format instead of : XML Format

```
response.then().contentType(ContentType.JSON);
```

```
Exception in thread "main" java.lang.AssertionError: 1 expectation failed.
Expected content-type "XML" doesn't match actual content-type "application/json; charset=utf-8".

	at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
	at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:77)
	at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:
	at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)
	at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:480)
	at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:72)
	at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConstructor(ConstructorSite
	at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:59)
	at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:263)
	at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:277)
	at io.restassured.internal.ResponseSpecificationImpl$HamcrestAssertionClosure.validate(ResponseSpecificationImpl.groovy
	at io.restassured.internal.ResponseSpecificationImpl$HamcrestAssertionClosure$validate$1.call(Unknown Source)
	at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.java:47)
	at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:125)
	at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:139)
	at io.restassured.internal.ResponseSpecificationImpl.validateResponseIfRequired(ResponseSpecificationImpl.groovy:684)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
	at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.base/java.lang.reflect.Method.invoke(Method.java:568)
	at org.codehaus.groovy.runtime.callsite.PlainObjectMetaMethodSite.doInvoke(PlainObjectMetaMethodSite.java:43)
	at org.codehaus.groovy.runtime.callsite.PogoMetaMethodSite$PogoCachedMethodSiteNoUnwrapNoCoerce.invoke(PogoMetaMethodSi
	at org.codehaus.groovy.runtime.callsite.PogoMetaMethodSite.callCurrent(PogoMetaMethodSite.java:61)
```

We can use another way to validate the content:

response.then().contentType("application/json");

```
response.then().contentType("application/xml");
response.then().contentType("application/json");
```

```java
import io.restassured.http.ContentType;

public class GetUserByID {

    public static void main(String[] args) {
        String url="https://gorest.co.in/public/v1/users/5231685";

        Response resp=given()
                            .header("Authorization","Bearer 1b0a651f0ee6151ed5586b6
                        .when()
                             .get(url);

        //resp.prettyPrint();

        //validate status code
        resp.then().statusCode(200);

        // get status code
        int statusCode=resp.getStatusCode();
        System.out.println(statusCode);

        //validate response data/content type
        //resp.then().contentType("application/json");
        resp.then().contentType(ContentType.JSON);


    }

}
```

Extract value from response

In order to retrieve an element value from json response, we can use **extract().path()** methods and need to provide path of that element.

Example: the path of "**total**" in this json data is:
meta.pagination.total


```
{
    "meta": {
        "pagination": {
            "total": 2526,
            "pages": 253,
            "page": 1,
            "limit": 10,
            "links": {
```


```java
Response response=given().when().get(url);

int total=response.then().extract().path("meta.pagination.total");
```

response.then().extract().path("meta.pagination.total") extracts the value of the total field from the JSON response.

- The JSON path `meta.pagination.total` specifies the location of the `total` field in the JSON structure.

Store the Extracted Value:

- The extracted value is stored in the `total` variable (in this case, an `int`).

Task :

Extract the email

You do not need to use : meta

Extract the id.

You need to use Integer for the id. The id does not have the double quote used for String

```
{
    "meta": null,
    "data": {
        "id": 5231685,
        "name": "Cohort11",
        "email": "elshanCohort34@gmail.com",
        "gender": "male",
        "status": "active"
    }
}
```

Result:

```
{
    "meta": null,
    "data": {
        "id": 5231685,
        "name": "Cohort11",
        "email": "elshanCohort34@gmail.com",
        "gender": "male",
        "status": "active"
    }
}
elshanCohort34@gmail.com
5231685
```

Task : Extract the total for active women

```
import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class GetUserList {

    public static void main(String[] args) {

          String url="https://gorest.co.in/public/v1/users";
         // submit API GET request and receive response
          Response response= given()
                                .param("gender", "female")
                                .param("status", "active")
                            .when()
                                .get(url);

         int totalUser= response.then().extract().path("meta.pagination.total");
         System.out.println(totalUser);
```

Result: 1040

```
1040
{
    "meta": {
        "pagination": {
            "total": 1040,
            "pages": 104,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    }
```

There are another way to retrieve Data !

See :

POST METHOD

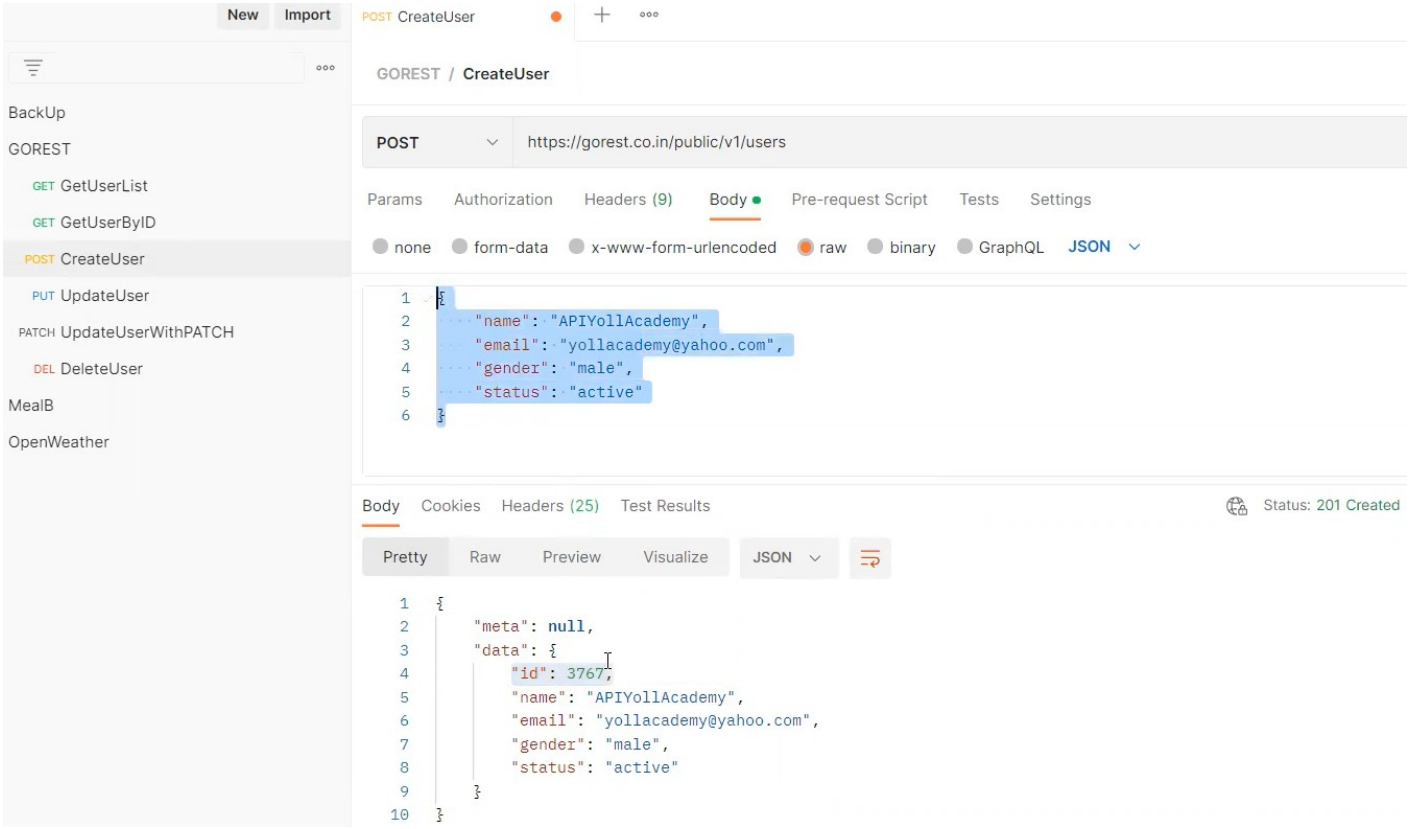
post() >>> method is used to create new record.

body() >>> method is used to send request data (json, xml , etc.).
It can be used in POST, PUT, PATCH request

contentType() >>> is used to specify the format of the request data we submit in **body()** method.
Where we use **body()** method, we should also use **contentType()** method.

```
String url="https://gorest.co.in/public/v1/users";

String jsonData="{\"name\": \"John Smith\",\"email\": \"john@gmail.com\","
                + "\"gender\": \"male\",\"status\": \"active\"}";

Response response=given()
                    .header("Authorization","Bearer a4kdsu4981")
                    .contentType(ContentType.JSON)
                    .body(jsonData)
                 .when()
                    .post(url);
```

In Postman we use Post:



By automation

```
public class CreateUser {

    public static void main(String[] args) {

            String url="https://gorest.co.in/public/v1/users";

            String requestData="{\r\n"
                    + "    \"name\": \"APIYollAcademy\",\r\n"
                    + "    \"email\": \"yollacademy123@yahoo.com\",\r\n"
                    + "    \"gender\": \"male\",\r\n"
                    + "    \"status\": \"active\"\r\n"
                    + "}";

        Response response= given()
                            .header("Authorization","Bearer 1b0a651f0ee6151ed5586b675cd871b4178
                            .body(requestData)
                            .contentType(ContentType.JSON)
                        .when()
                            .post(url);

        response.prettyPrint();

        int statusCode=response.getStatusCode();
        System.out.println(statusCode);
```

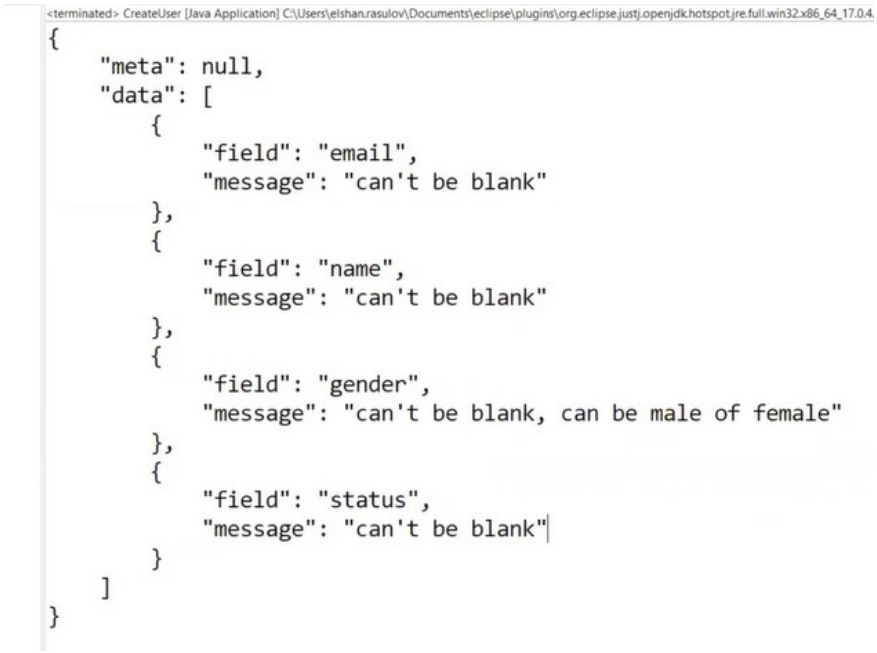
We put the variables (id, name, email, gender, status) inside the body.

Using contentType is very important to print JSON Format. Otherwise, it will not be printed !

```
package gorest;

import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class CreateUser {

    public static void main(String[] args) {
        // API endpoint for creating a user
        String url = "https://gorest.co.in/public/v1/users";

        // JSON payload for the request
        String requestData = "{"
                + "\"name\": \"APIVO11Academy\","
                + "\"email\": \"yollacademy123@yahoo.com\","
                + "\"gender\": \"male\","
                + "\"status\": \"active\""
                + "}";
```

```
        // Send POST request with authorization header and JSON payload
        Response response = given()
                .header("Authorization", "Bearer YOUR_ACCESS_TOKEN_HERE") // Replace with your actual token
                .contentType(ContentType.JSON)
                .body(requestData)
                .post(url);

        // Print the response status code and body
        System.out.println("Status Code: " + response.getStatusCode());
        System.out.println("Response Body: " + response.prettyPrint());
    }
}
```

The Result if `contentType(ContentType.JSON)` is not used:



The Result After using `contentType(ContentType.JSON)`:

***Post / Create a user

@Test

With @Test Annotation

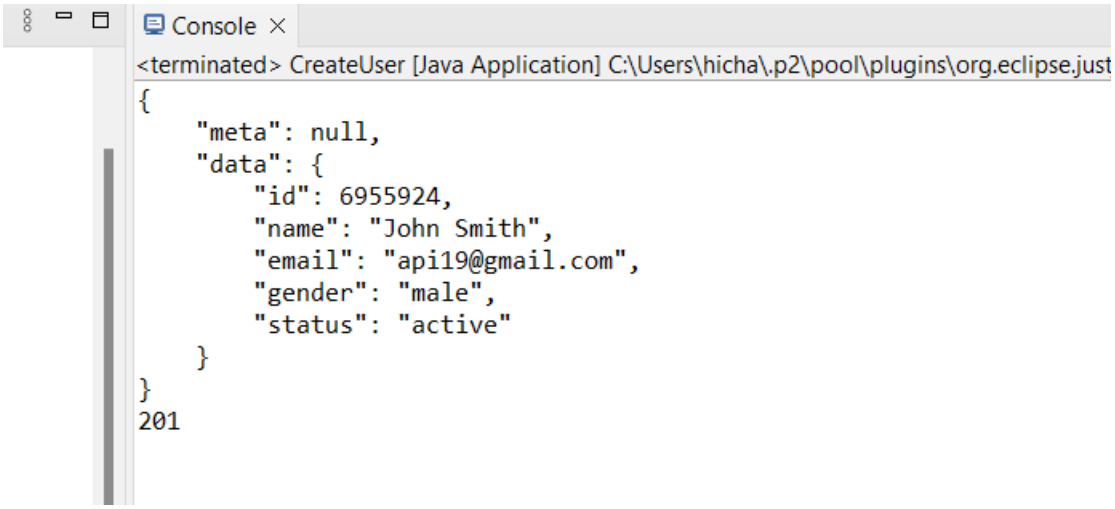
```
public class CreateUser {

    @Test
    public void addGorestUser() {
        String url="https://gorest.co.in/public/v1/users";
        String token="Bearer 1b0a651f0ee6151ed5586b675cd871b4178f23f1ee6900f0fb7a34d787ac1855";
        String payload="{\r\n"
                + "    \"name\": \"Cohort12\",\r\n"
                + "    \"email\": \"abcde1234@gmail.com\",\r\n"
                + "    \"gender\": \"male\",\r\n"
                + "    \"status\": \"active\"\r\n"
                + "}";

        Response response=given()
                            .header("Authorization",token)
                            //.header("Content-Type", "application/json")
                            .contentType(ContentType.JSON)
                            .body(payload)
                        .when()
                            .post(url);

        response.prettyPrint();

        System.out.println(response.getStatusCode());


    }
}
```

```
package gorest;

import org.junit.Test;
import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class CreateUserJUNIT {
      @Test
      public void addgorestUser(){
            String url="https://gorest.co.in/public/v1/users";
            String token="Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";
            String payload="{\r\n"
                        + "    \"name\": \" Cohort12\",\r\n"
                        + "    \"email\": \"abcd123@gmail.com\",\r\n"
                        + "    \"gender\": \"male\",\r\n"
                        + "    \"status\": \"active\"\r\n"
                        + "}";
            Response response=given()
                        .header("Authorization",token)
                        .header("Content-Type","application/json")
```

```
                        .body(payload)
                        .when()
                        .post(url);
            response.prettyPrint();

                        System.out.println(response.getStatusCode());

      }
      }
```

Without @Test Annotation

```
package gorest;
import static io.restassured.RestAssured.given;

//import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class CreateUser {

      public static void main(String[] args) {
            String url="https://gorest.co.in/public/v1/users";
            String requestData=" {\r\n"
                        + "\"name\": \"John Smith\",\r\n"
                        + "\"email\": \"api19@gmail.com\",\r\n"
                        + "\"gender\": \"male\",\r\n"
                        + "\"status\": \"active\"\r\n"
                        + "}";
Response response=given()
.header("Authorization","Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
            //we can put the Content Type on Header as Postman
            .body(requestData)
            .header("Content-Type","application/json")
            //Second way:
            //.contentType(ContentType.JSON)
            .when()
            .post(url);
            response.prettyPrint();

            //Get status code => it returns integer
                        int statusCode=response.getStatusCode();
                        System.out.println(statusCode);

      }

}
```



This Java code uses the `RestAssured` library for making HTTP requests and `JUnit` for writing test cases. It demonstrates how to create a new user on the GoRest API. Here is a detailed explanation of each part of the code:

Package Declaration

```
package gorest;
```

- Declares the package name `gorest`.

Imports

```
import org.junit.Test;
import static io.restassured.RestAssured.given;
import io.restassured.response.Response;
```

- Imports necessary classes:
  - `org.junit.Test`: Used for writing test methods in JUnit.
  - `io.restassured.RestAssured.given`: Provides a static import for setting up the request.
  - `io.restassured.response.Response`: Used to handle the response from the API.

Class Declaration

```
public class CreateUserJUNIT {
```

- Declares the class `CreateUserJUNIT`.

Test Method

```
@Test
public void addgorestUser(){
```

- Annotates the method `addgorestUser` with `@Test`, indicating that this is a test case to be run by JUnit.

URL, Token, and Payload

```
String url="https://gorest.co.in/public/v1/users";
String token="Bearer 3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";
String payload="{\r\n"
        + "    \"name\": \" Cohort12\",\r\n"
        + "    \"email\": \"abcd123@gmail.com\",\r\n"
        + "    \"gender\": \"male\",\r\n"
        + "    \"status\": \"active\"\r\n"
        + "}";
```

- `url`: The endpoint for creating a new user on the GoRest API.
- `token`: The Bearer token for authorization.
- `payload`: The JSON string representing the user data to be sent in the request body.

Making the HTTP POST Request

```
Response response=given()
        .header("Authorization",token)
        .header("Content-Type","application/json")
        .body(payload)
        .when()
        .post(url);
```

- `given()`: Begins the request specification.
- `.header("Authorization", token)`: Adds the `Authorization` header with the Bearer token.
- `.header("Content-Type", "application/json")`: Sets the `Content-Type` header to `application/json`.
- `.body(payload)`: Sets the request body to the user data defined in `payload`.
- `.when()`: Specifies that the following method call will execute the request.
- `.post(url)`: Sends a POST request to the specified `url`.

Printing the Response

```
response.prettyPrint();
```

- `response.prettyPrint()`: Prints the response body in a pretty format.

Printing the Status Code

```
System.out.println(response.getStatusCode());
```

- `response.getStatusCode()`: Retrieves the status code from the response and prints it to the console.

Closing the Class

```
}
```

- Closes the `addgorestUser` method and the `CreateUserJUNIT` class.

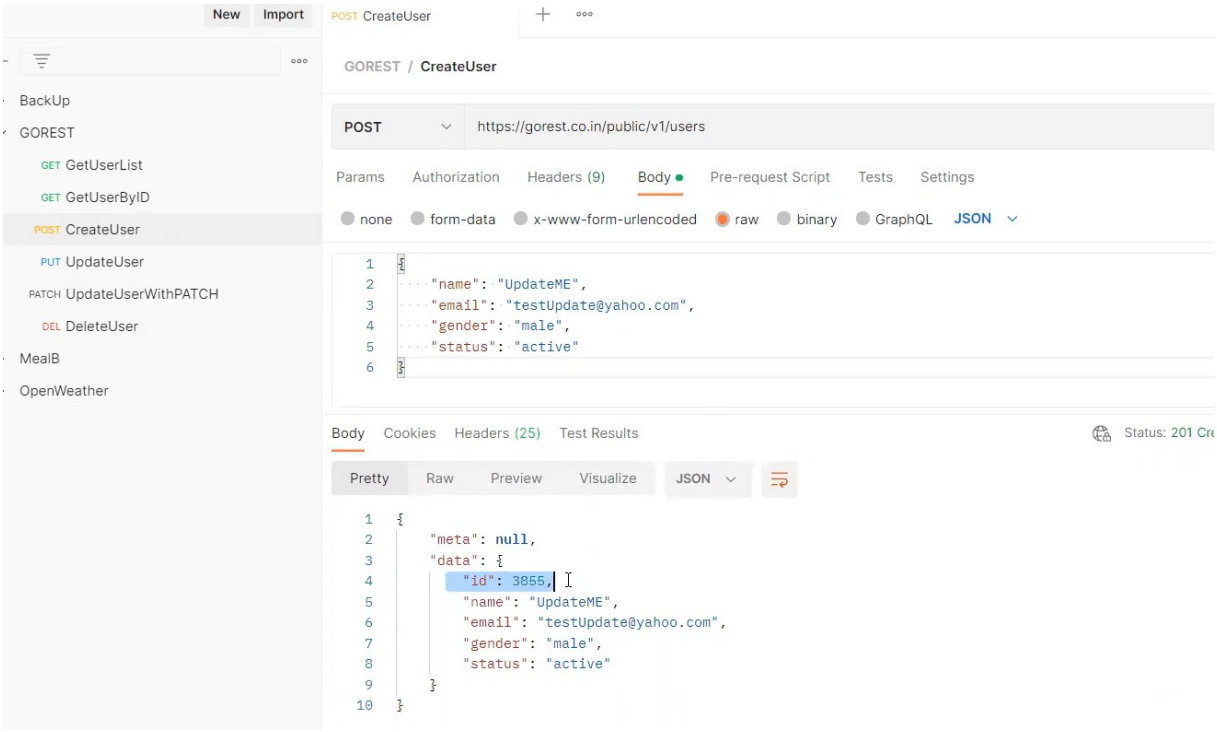
Summary

This code performs the following actions:

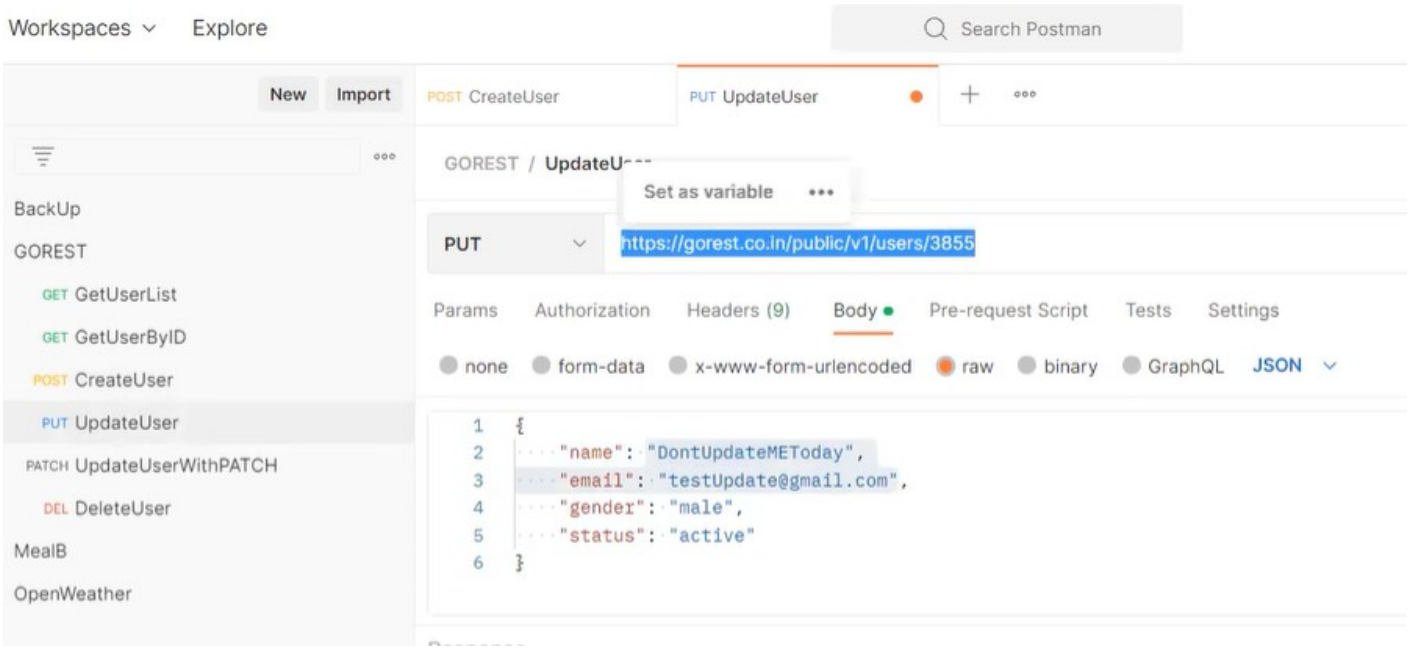
1. Defines the necessary URL, authorization token, and JSON payload for creating a user.
2. Sends a POST request to the GoRest API with the provided data.
3. Prints the response body and status code to the console.

PUT Method / Update a user

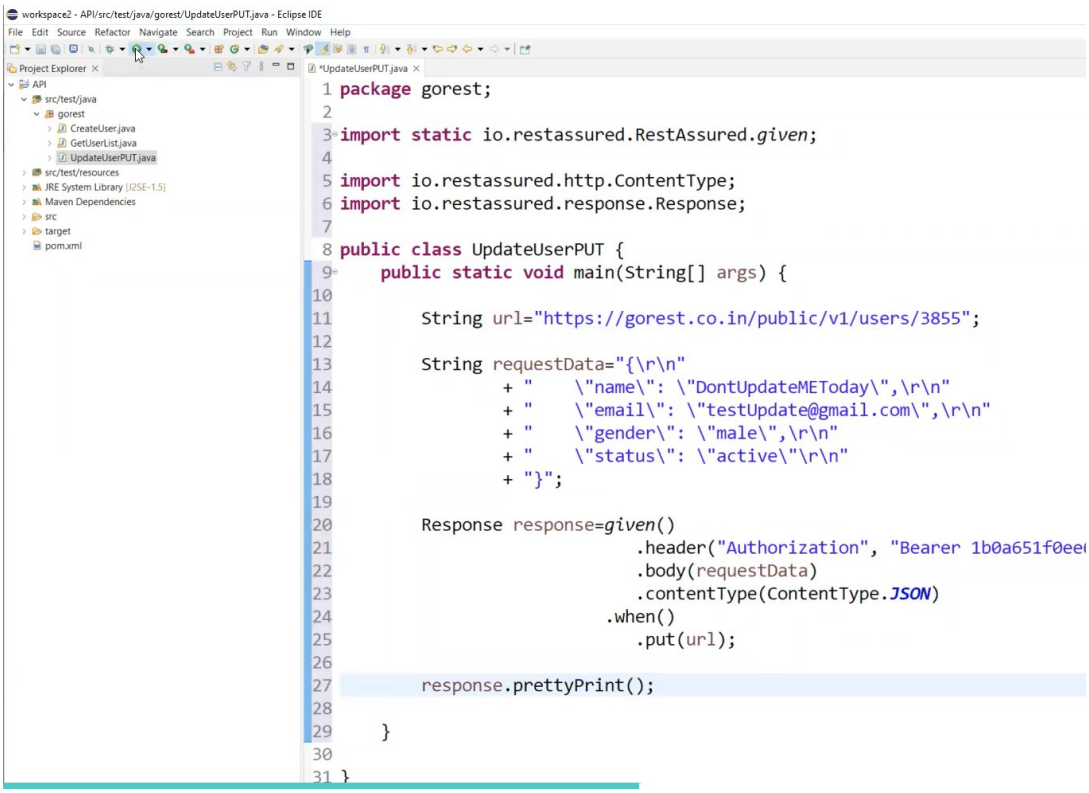
Before using Put Method, let’s create a new user on Postman bu using Post Method:



Change the first two elements: name and email



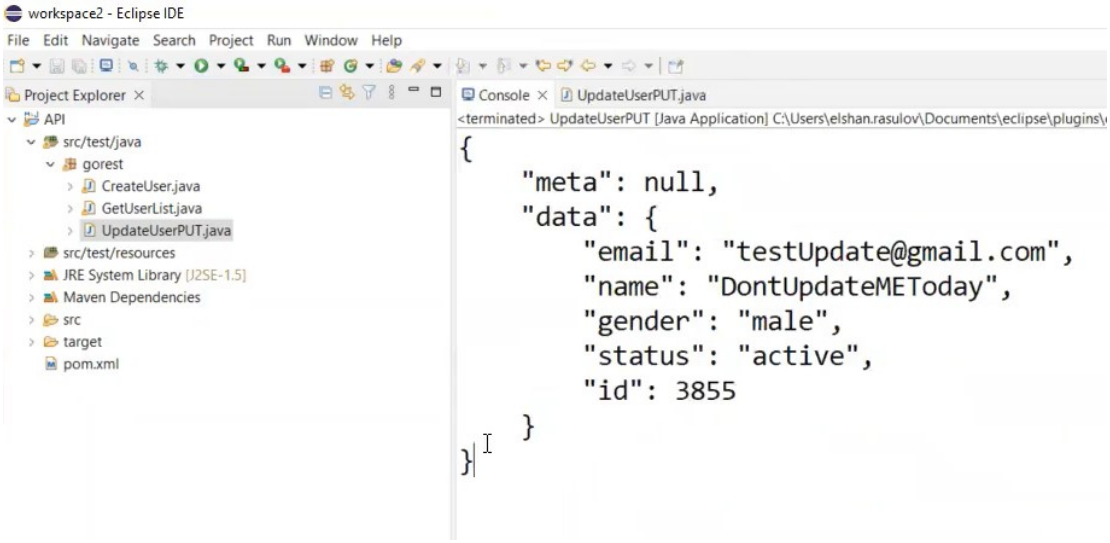
By automation:



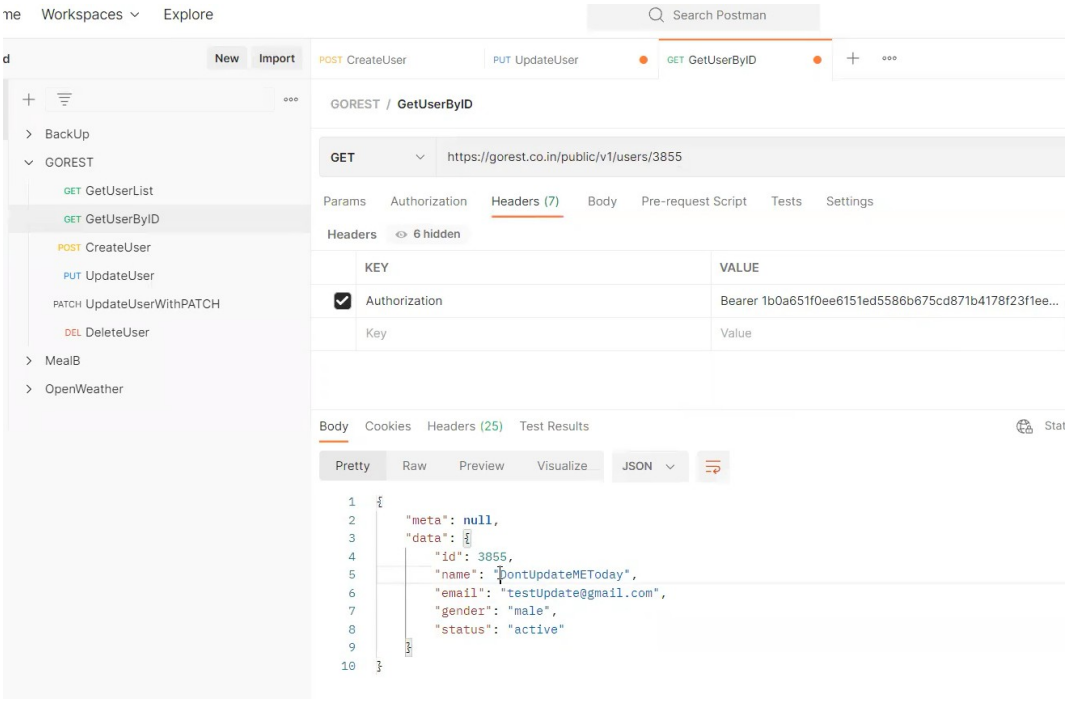

```java
package gorest;
import static io.restassured.RestAssured.given;

import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class UpdateUserPut {
	public static void main(String[] args) {

		String url="https://gorest.co.in/public/v1/users/3855";
		String requestData=" {\r\n"
```

```
                        + "        \"name\": \"Somi Nour\",\r\n"
                        + "        \"email\": \"nair_somu@adams.example\",\r\n"
                        + "        \"gender\": \"male\",\r\n"
                        + "        \"status\": \"inactive\"\r\n"
                        + "    }";
Response response=given()
        .header("Authorization","Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
            //we can put the Content Type on Header as Postman
            .body(requestData)
            .header("Content-Type","application/json")
            //Second way:
            //.contentType(ContentType.JSON)
            .when()
            .put(url);
            response.prettyPrint();

            //Get status code => it returns integer
                        int statusCode=response.getStatusCode();
                        System.out.println(statusCode);


      }

}
```

Result after run it: the two elements name and email are updated successfully for the user : 3855

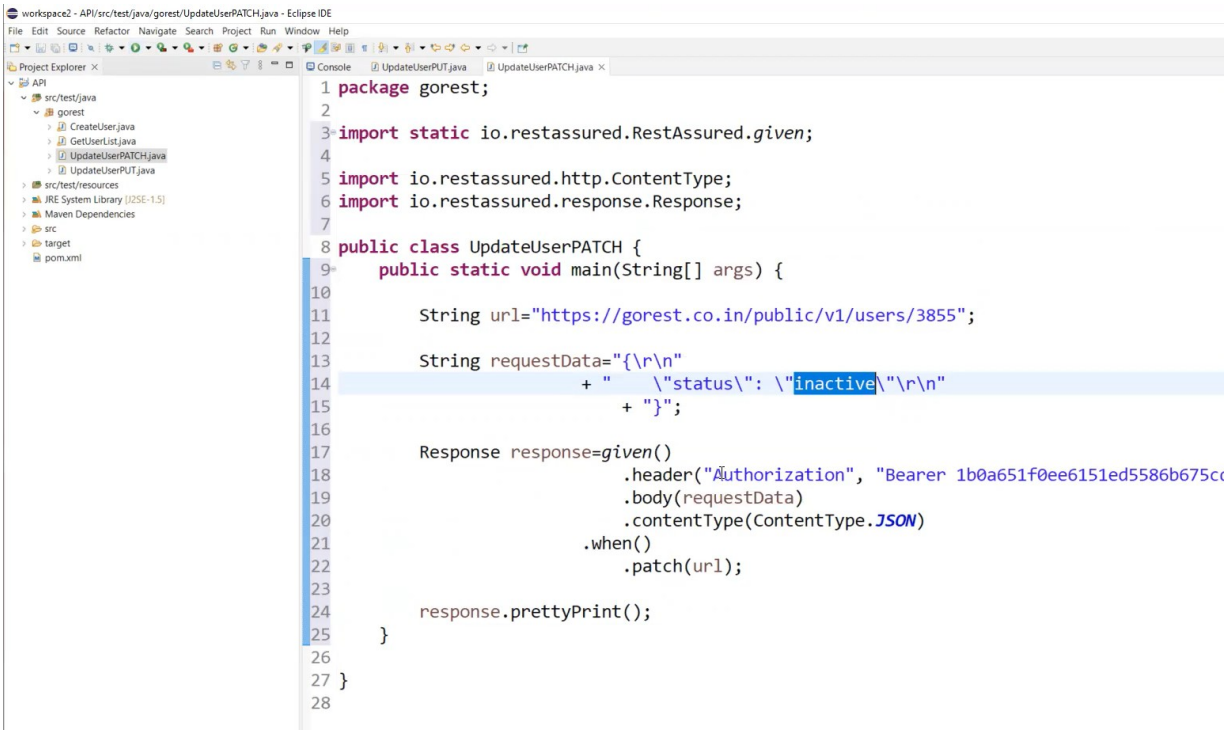


Result on Postman by using Get Method for the user 3855



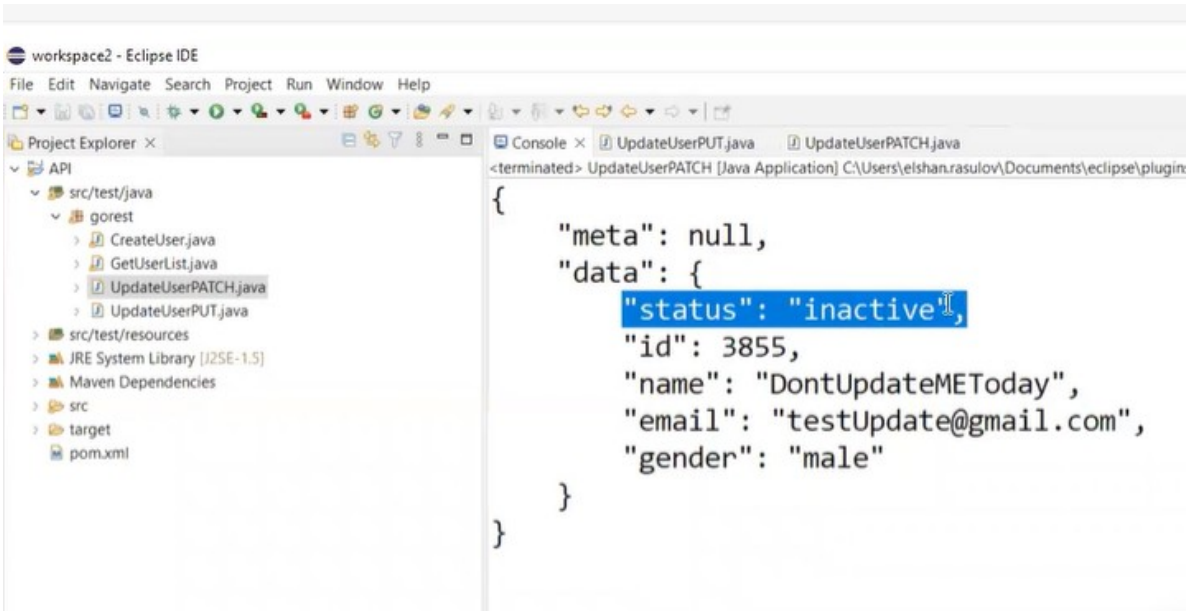
Patch Method:

We need to change only the status from active to inactive for the user :3855, and do not to change the other elements (gender, name, email)



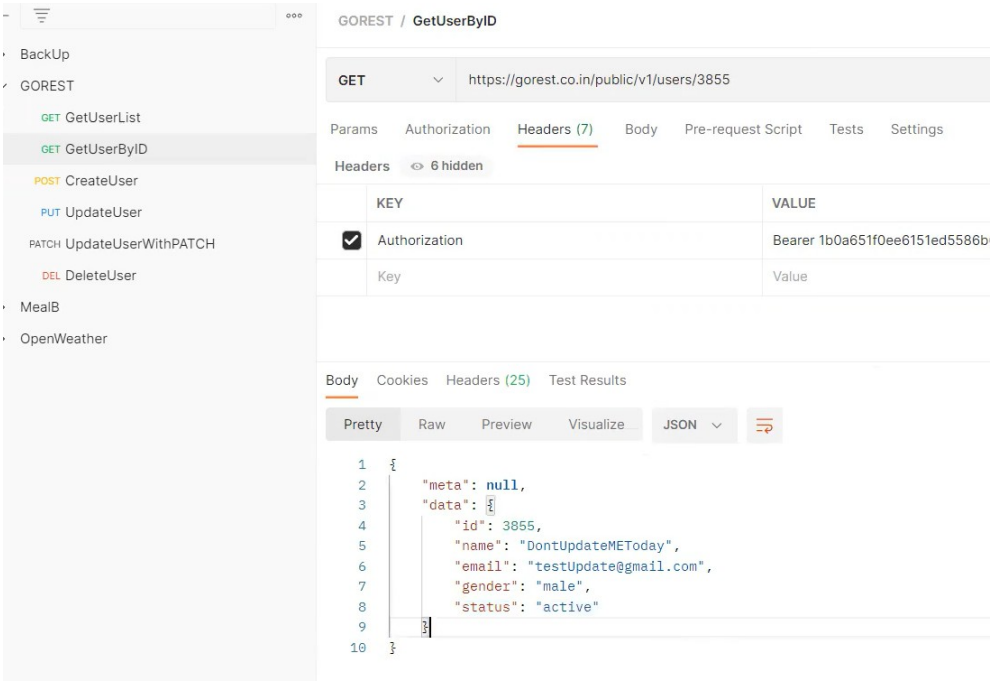
```java
package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class UpdateUserPATCH {
public static void main(String[] args) {

            String url="https://gorest.co.in/public/v1/users/3855";
            String requestData=" {\r\n"

                        + "        \"status\": \"inactive\"\r\n"
                        + "    }";
Response response=given()
        .header("Authorization","Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
            //we can put the Content Type on Header as Postman
            .body(requestData)
            .header("Content-Type","application/json")
            //Second way:
            //.contentType(ContentType.JSON)
            .when()
            .patch(url);
            response.prettyPrint();

            //Get status code => it returns integer
                        int statusCode=response.getStatusCode();
                        System.out.println(statusCode);


      }

}
```

Result:

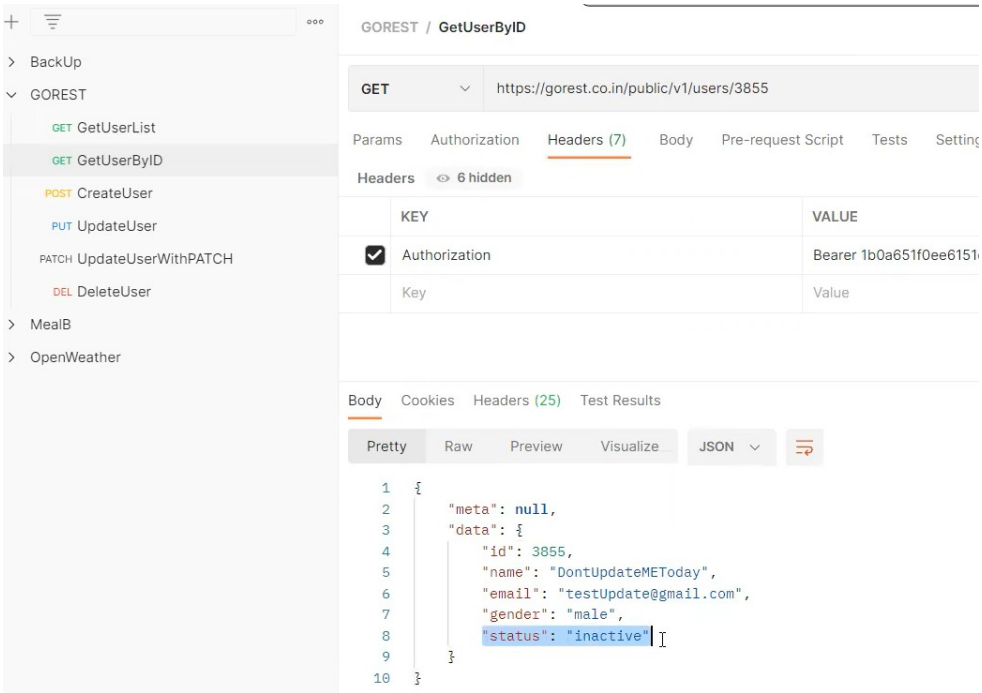


Let’s see the result on Postman:

Before using Patch Method: the status was active



After using Patch Method: the status became inactive

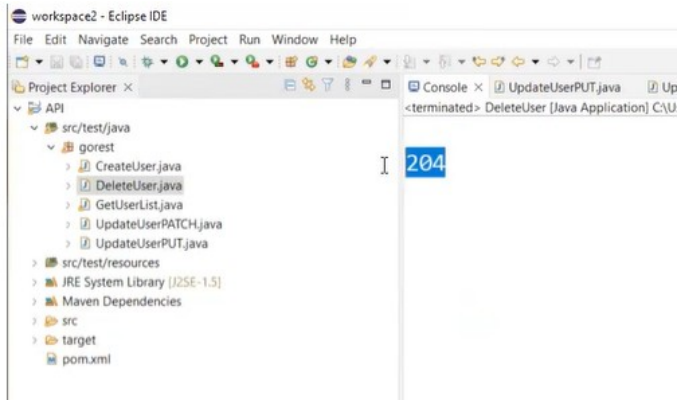


Delete Method:

Delete the user : 3855.

We do not see any data like post, get, and put.

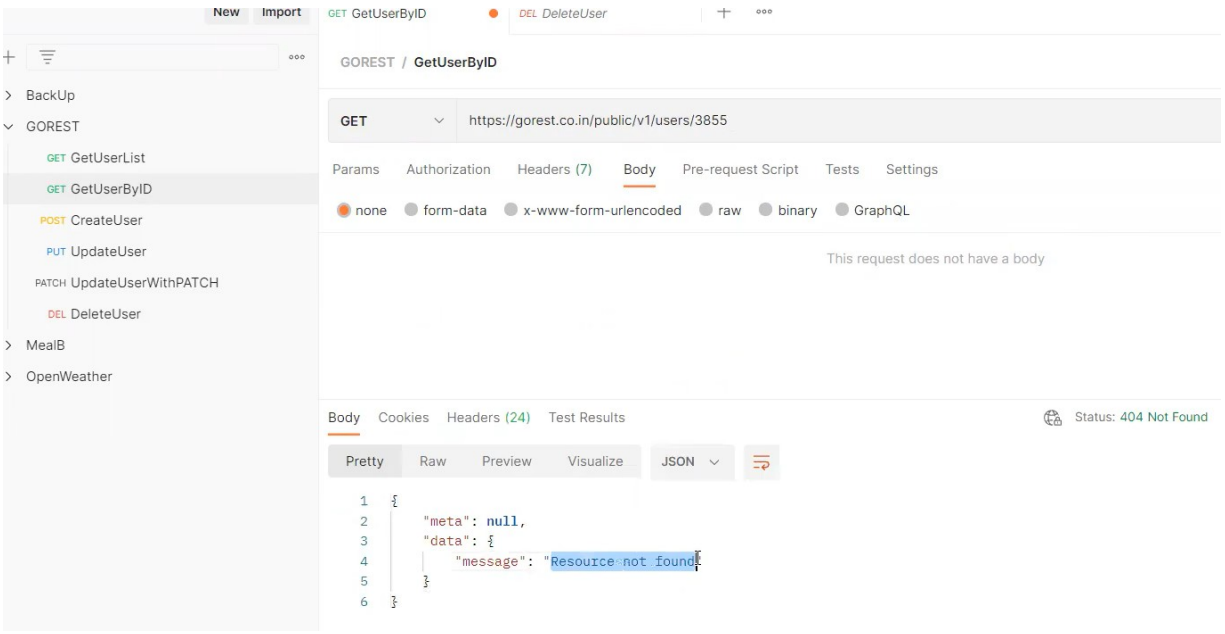
```
package gorest;

import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class DeleteUser {
public static void main(String[] args) {

            String url="https://gorest.co.in/public/v1/users/3855";

Response response=given()
        .header("Authorization","Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
            .when()
            .delete(url);
            response.prettyPrint();

            //Get status code => it returns integer
                        int statusCode=response.getStatusCode();
                        System.out.println(statusCode);


      }
}
```

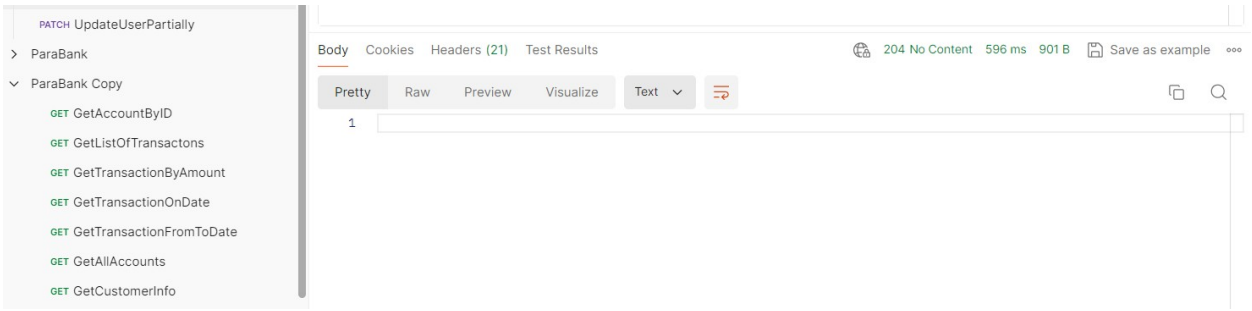


Status: 204 No Content **Because it was deleted !**

To check if the user is deleted, we use Get Method for the user:3855.

After sending the request, there is no user found which means it was deleted successfully.

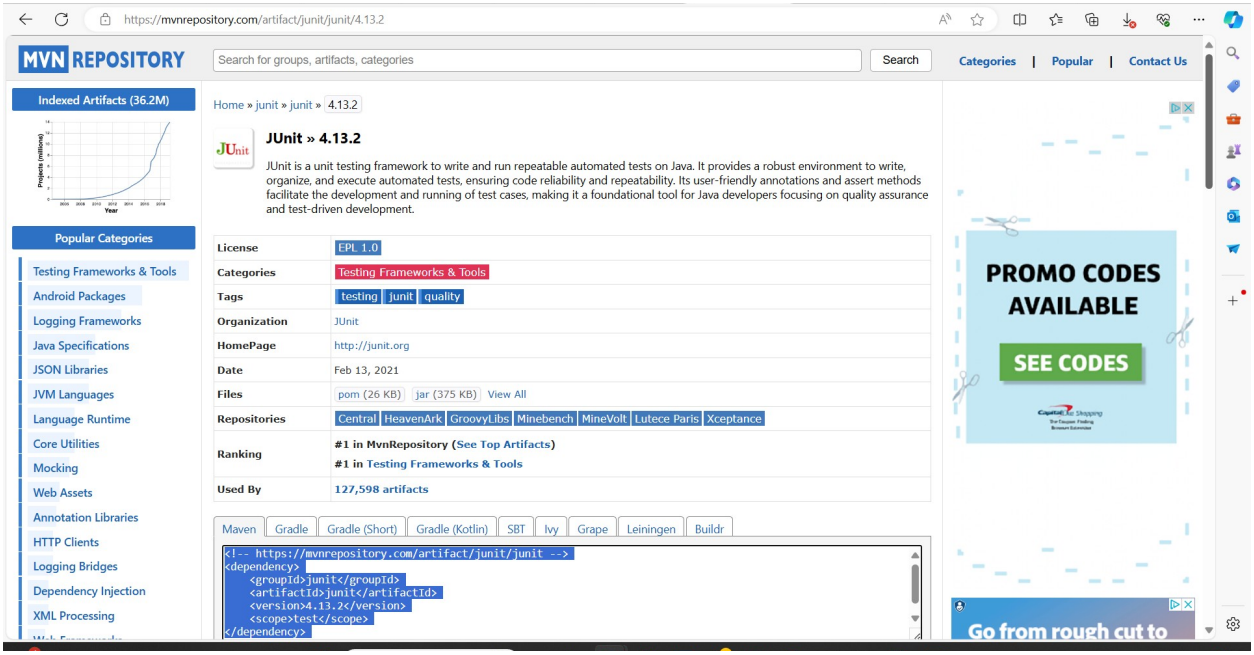

New
Import
GET GetUserByID
DEL DeleteUser
BackUp
GOREST
GET GetUserList
GET GetUserByID
POST CreateUser
PUT UpdateUser
PATCH UpdateUserWithPATCH
DEL DeleteUser
MealB
OpenWeather
GOREST / GetUserByID
GET
https://gorest.co.in/public/v1/users/3855
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body
Body
Cookies
Headers (24)
Test Results
Status: 404 Not Found
Pretty
Raw
Preview
Visualize
JSON
{
"meta": null,
"data": {
"message": "Resource not found"
}
}


PATCH UpdateUserPartially
ParaBank
ParaBank Copy
GET GetAccountByID
GET GetListOfTransactons
GET GetTransactionByAmount
GET GetTransactionOnDate
GET GetTransactionFromToDate
GET GetAllAccounts
GET GetCustomerInfo
Body
Cookies
Headers (21)
Test Results
204 No Content
596 ms
901 B
Save as example
Pretty
Raw
Preview
Visualize
Text
1

Automation by using Junit

Put Junit dependency on pom.xml

Add @ Annotation before the methods.


https://mvnrepository.com/artifact/junit/junit/4.13.2
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (36.2M)
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Language Runtime
Core Utilities
Mocking
Web Assets
Annotation Libraries
HTTP Clients
Logging Bridges
Dependency Injection
XML Processing
Home » junit » junit » 4.13.2
JUnit » 4.13.2
JUnit is a unit testing framework to write and run repeatable automated tests on Java. It provides a robust environment to write, organize, and execute automated tests, ensuring code reliability and repeatability. Its user-friendly annotations and assert methods facilitate the development and running of test cases, making it a foundational tool for Java developers focusing on quality assurance and test-driven development.
License EPL 1.0
Categories Testing Frameworks & Tools
Tags testing junit quality
Organization JUnit
HomePage http://junit.org
Date Feb 13, 2021
Files pom (26 KB) jar (375 KB) View All
Repositories Central HeavenArk GroovyLibs Minebench MineVolt Lutece Paris Xceptance
Ranking #1 in MvnRepository (See Top Artifacts)
#1 in Testing Frameworks & Tools
Used By 127,598 artifacts
Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.13.2</version>
<scope>test</scope>
</dependency>
PROMO CODES AVAILABLE
SEE CODES
Go from rough cut to

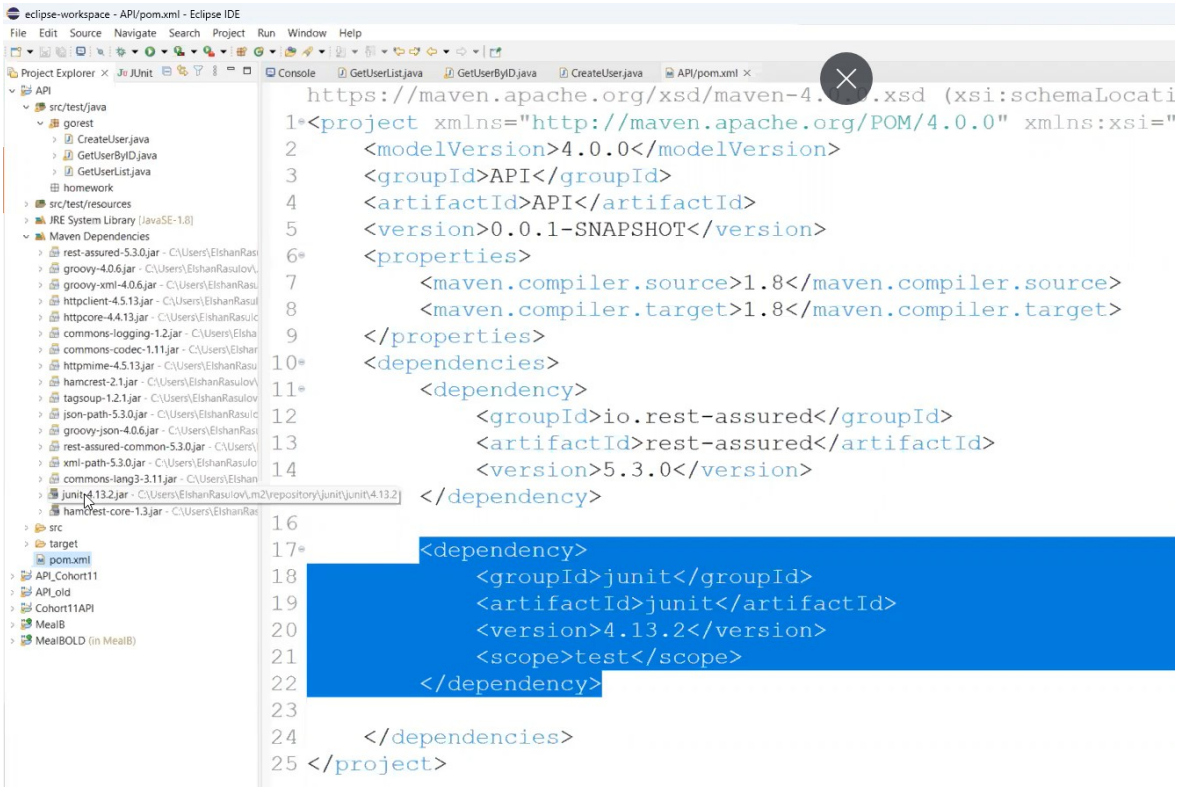
```
<!-- https://mvnrepository.com/artifact/junit/junit -->
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>4.13.2</version>
			<scope>test</scope>
		</dependency>
```

GET_Request.java | resquestData.json | *REST_API/pom.xml

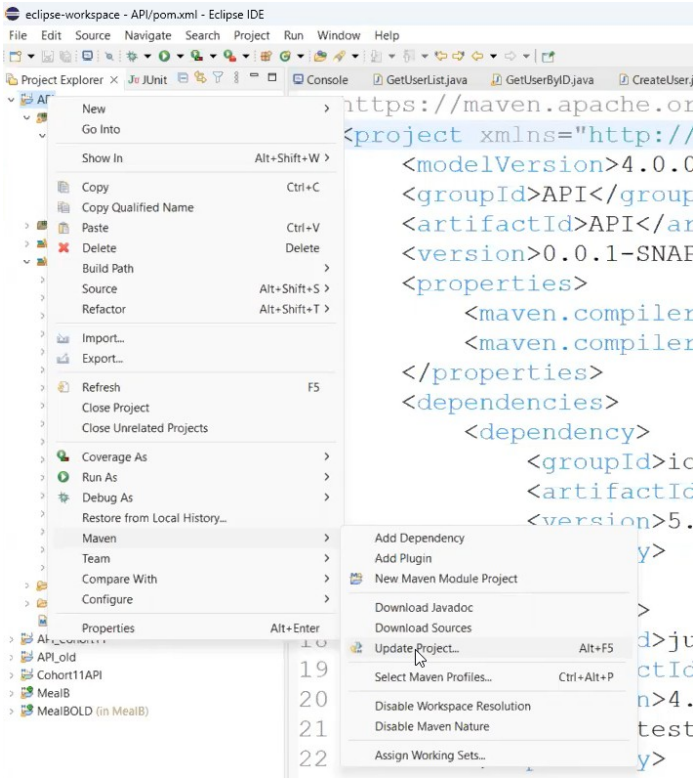
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xs
    <modelVersion>4.0.0</modelVersion>
    <groupId>yoll.io</groupId>
    <artifactId>REST_API</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>

        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>5.2.0</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>

```

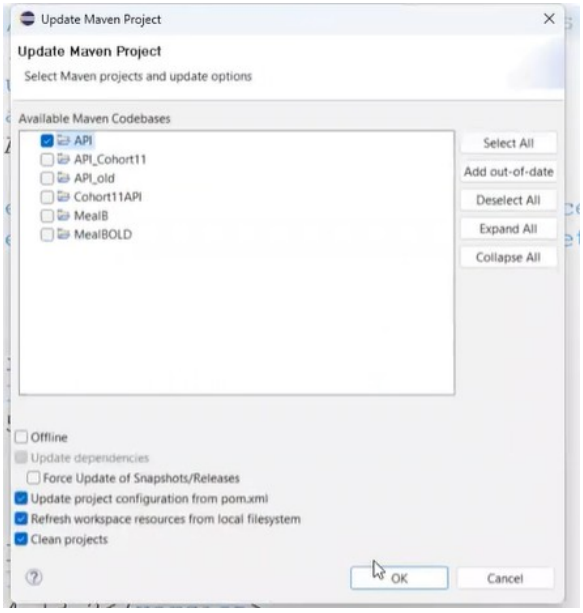
You have to see on maven dependencies : JUNIT-4.132 JAR File

Do not forget to save the file !



Update the Project if you do not see the Jar files


eclipse-workspace - API/pom.xml - Eclipse IDE
File Edit Source Navigate Search Project Run Window Help
Project Explorer
JUnit
Console
GetUserList.java
GetUserByID.java
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Coverage As
Run As
Debug As
Restore from Local History...
Maven
Team
Compare With
Configure
Properties Alt+Enter
Add Dependency
Add Plugin
New Maven Module Project
Download Javadoc
Download Sources
Update Project... Alt+F5
Select Maven Profiles... Ctrl+Alt+P
Disable Workspace Resolution
Disable Maven Nature
Assign Working Sets...
API_old
Cohort11API
MealB
MealBOLD (in MealB)
https://maven.apache.or
<project xmlns="http://
<modelVersion>4.0.0
<groupId>API</group
<artifactId>API</ar
<version>0.0.1-SNAP
<properties>
<maven.compiler
<maven.compiler
</properties>
<dependencies>
<dependency>
<groupId>ic
<artifactId
<version>5.


Update Maven Project
Update Maven Project
Select Maven projects and update options
Available Maven Codebases
API
API_Cohort11
API_old
Cohort11API
MealB
MealBOLD
Select All
Add out-of-date
Deselect All
Expand All
Collapse All
Offline
Update dependencies
Force Update of Snapshots/Releases
Update project configuration from pom.xml
Refresh workspace resources from local filesystem
Clean projects
OK
Cancel

In **JUnit**, annotations are used to define and configure test methods, lifecycle methods, and other behaviors. Below is a detailed explanation of the most commonly used JUnit annotations, including @Test, and how they are used in automated testing.

---

1. Core Annotations

@Test

- Marks a method as a **test method**.
- JUnit will execute this method as part of the test suite.
- Example:

```
@Test
public void testMethod() {
    assertEquals(2, 1 + 1, "1 + 1 should equal 2");
}
```

@BeforeEach

- Marks a method to be executed **before each test method**.
- Used for setup tasks (e.g., initializing objects, setting up test data).
- Example:

```
@BeforeEach
public void setUp() {
    System.out.println("Setting up before each test");
}
```

@AfterEach

- Marks a method to be executed **after each test method**.
- Used for cleanup tasks (e.g., releasing resources, resetting states).
- Example:

```
@AfterEach
public void tearDown() {
    System.out.println("Cleaning up after each test");
}
```

@BeforeAll

- Marks a method to be executed **once before all test methods** in the class.
- The method must be static.

- Used for global setup tasks (e.g., initializing shared resources).
- Example:

```
@BeforeAll
public static void init() {
    System.out.println("Initializing before all tests");
}
```

@AfterAll

- Marks a method to be executed **once after all test methods** in the class.
- The method must be static.
- Used for global cleanup tasks (e.g., closing database connections).
- Example:

```
@AfterAll
public static void cleanup() {
    System.out.println("Cleaning up after all tests");
}
```

---

2. Advanced Annotations

@DisplayName

- Provides a custom name for the test class or method.
- Useful for making test reports more readable.
- Example:

```
@Test
@DisplayName("Test addition of two numbers")
public void testAddition() {
    assertEquals(3, 1 + 2, "1 + 2 should equal 3");
}
```

@Disabled

- Disables a test method or class.
- Useful for temporarily skipping tests that are not ready or no longer relevant.
- Example:

```
@Test
@Disabled("This test is not implemented yet")
public void testNotImplemented() {
    // Test logic here
}
```

@ParameterizedTest

- Marks a method as a **parameterized test**.
- Allows running the same test with different inputs.
- Requires a source of arguments (e.g., @ValueSource, @CsvSource).
- Example:

```
@ParameterizedTest
@ValueSource(ints = {1, 2, 3})
public void testParameterized(int number) {
    assertTrue(number > 0, "Number should be positive");
}
```

@RepeatedTest

- Repeats a test a specified number of times.
- Useful for testing flaky or non-deterministic behavior.
- Example:

```
@RepeatedTest(5)
public void testRepeated() {
    assertNotNull(new Object(), "Object should not be null");
}
```

@Nested

- Marks a class as a **nested test class**.
- Allows grouping related tests together.
- Example:

```
@Nested
class NestedTests {
    @Test
    void nestedTest() {
        assertEquals(2, 1 + 1, "1 + 1 should equal 2");
    }
}
```

@Tag

- Adds a tag to a test method or class.
- Useful for filtering tests (e.g., running only specific groups of tests).
- Example:

```
@Test
@Tag("smoke")
public void smokeTest() {
```

```
    assertTrue(true, "This is a smoke test");
}
```

3. Argument Sources for Parameterized Tests

@ValueSource

- Provides a single array of values for parameterized tests.
- Example:

```
@ParameterizedTest
@ValueSource(strings = {"apple", "banana", "cherry"})
public void testFruits(String fruit) {
    assertNotNull(fruit, "Fruit should not be null");
}
```

@CsvSource

- Provides a list of comma-separated values (CSV) for parameterized tests.
- Example:

```
@ParameterizedTest
@CsvSource({"1, 2, 3", "4, 5, 9", "10, 20, 30"})
public void testAddition(int a, int b, int expected) {
    assertEquals(expected, a + b, "Addition should match");
}
```

@MethodSource

- Provides arguments from a factory method.
- Example:

```
@ParameterizedTest
@MethodSource("provideNumbers")
public void testNumbers(int number) {
    assertTrue(number > 0, "Number should be positive");
}

static Stream<Integer> provideNumbers() {
    return Stream.of(1, 2, 3);
}
```

Example of a Complete Test Class

```java
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

public class ExampleTest {

    @BeforeAll
    static void initAll() {
        System.out.println("Initializing before all tests");
    }

    @BeforeEach
    void init() {
        System.out.println("Setting up before each test");
    }

    @Test
    @DisplayName("Test addition")
    void testAddition() {
        assertEquals(4, 2 + 2, "2 + 2 should equal 4");
    }

    @Test
    @Disabled("This test is disabled")
    void disabledTest() {
        fail("This test should not run");
    }

    @AfterEach
    void tearDown() {
        System.out.println("Cleaning up after each test");
    }

    @AfterAll
    static void tearDownAll() {
        System.out.println("Cleaning up after all tests");
    }
}
```

Running Tests

- **From IDE**: Right-click the test class or method and select **Run**.
- **From Command Line**:

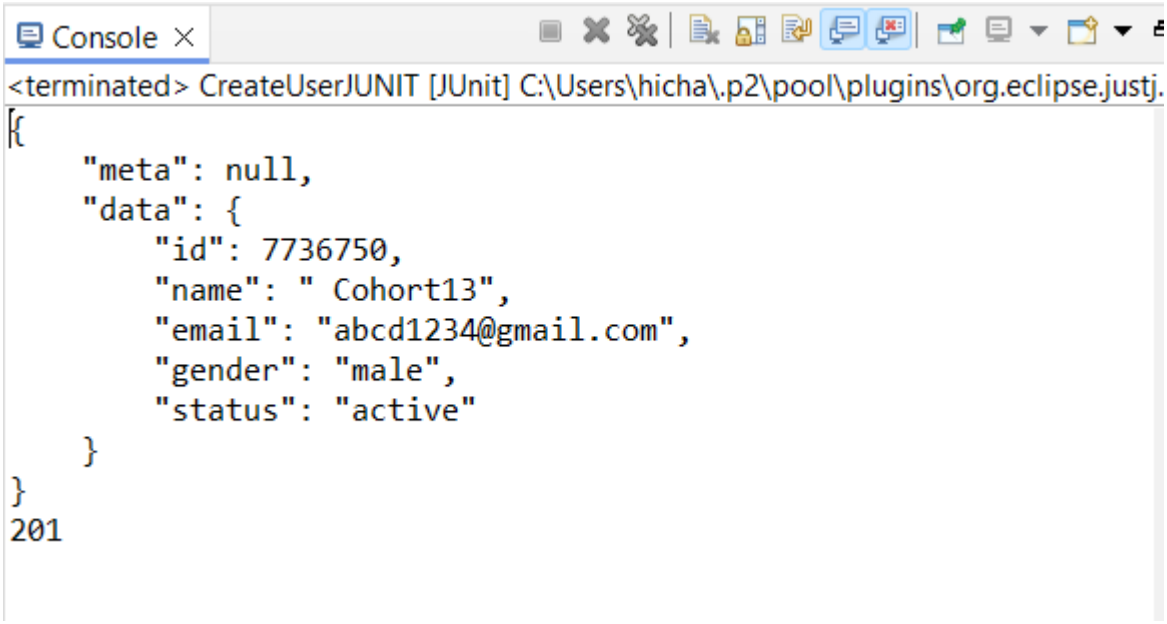
- Maven: mvn test
- Gradle: gradle tes

Adding @Test Annotation :

Write @Test, then add the library

POST Method Using @Test Annotation

```
package gorest;

import org.junit.Test;
import static io.restassured.RestAssured.given;

import io.restassured.response.Response;

public class CreateUserJUNIT {
	@Test
	public void addgorestUser(){
		String url="https://gorest.co.in/public/v1/users";
		String token="Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";
		String payload="{\r\n"
				+ "    \"name\": \" Cohort12\",\r\n"
				+ "    \"email\": \"abcd1234@gmail.com\",\r\n"
				+ "    \"gender\": \"male\",\r\n"
				+ "    \"status\": \"active\"\r\n"
				+ "}";
		Response response=given()
				.header("Authorization",token)
				.header("Content-Type","application/json")

				.body(payload)
				.when()
				.post(url);
		response.prettyPrint();

				System.out.println(response.getStatusCode());

	}
	}
```



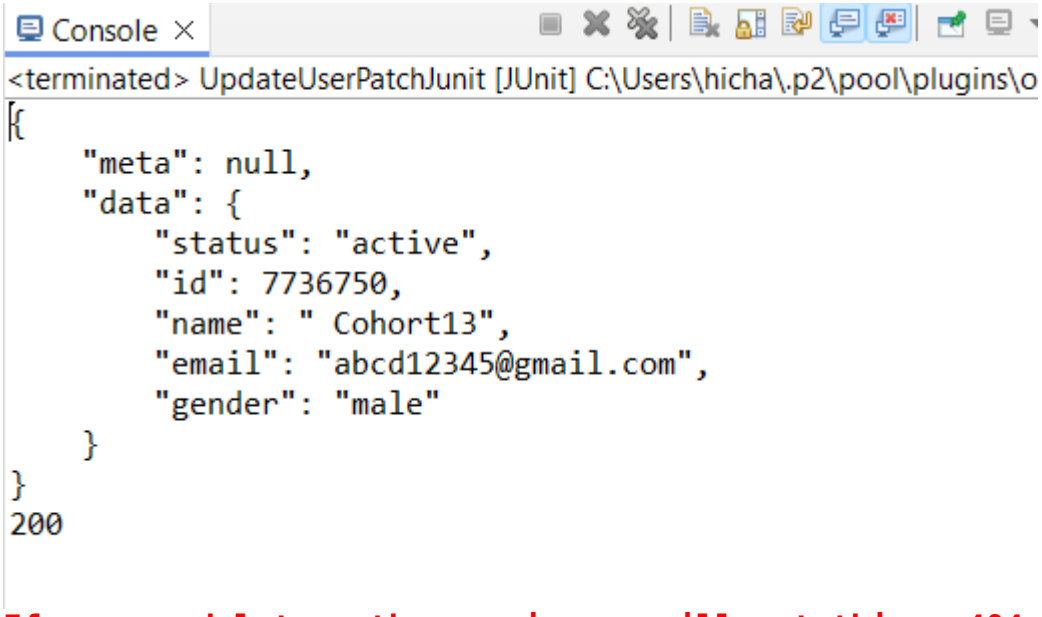
PUT Method Using @Test Annotation for user 7736750

```
package gorest;

import static io.restassured.RestAssured.given;
import org.junit.Test;
import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class UpdateUserJunit {

            @Test
            public void updateGorestUser(){
            String url="https://gorest.co.in/public/v1/users/";
            String token="Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";
            String requestData=" {\r\n"
                        + "    \"name\": \" Cohort13\",\r\n"
                        + "    \"email\": \"abcd12345@gmail.com\",\r\n"
                        + "    \"gender\": \"male\",\r\n"
                        + "    \"status\": \"INactive\"\r\n"
                        + "    }";
Response response=given()
        .header("Authorization",token)
            //we can put the Content Type on Header as Postman
            .body(requestData)
            .header("Content-Type","application/json")
            //Second way:
            //.contentType(ContentType.JSON)
            .when()
            .put(url+7736750);
            response.prettyPrint();

            //Get status code => it returns integer
                        int statusCode=response.getStatusCode();
                        System.out.println(statusCode);


      }
```

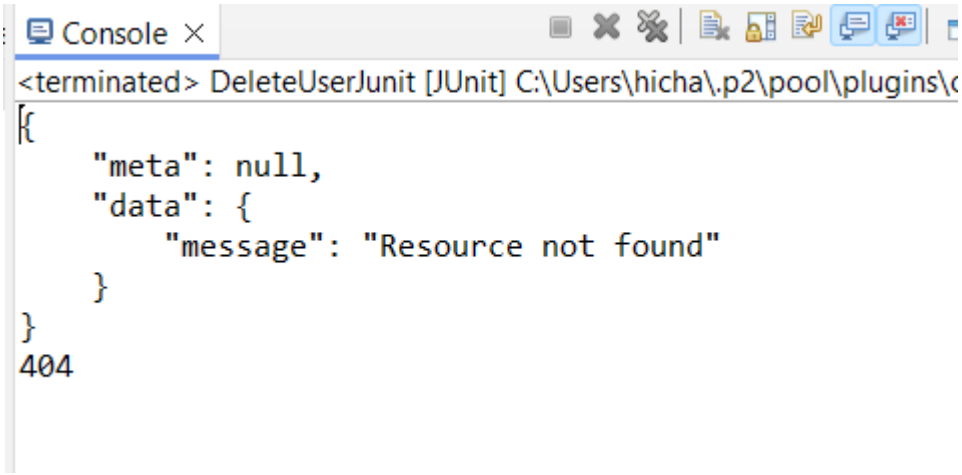
```
}
```

Patch Method Using @Test Annotation for user : **7736750**

```
package gorest;

import static io.restassured.RestAssured.given;
import org.junit.Test;
import io.restassured.http.ContentType;
import io.restassured.response.Response;

public class UpdateUserPatchJunit {
	@Test
	public void patchuser() {
		String url="https://gorest.co.in/public/v1/users/";
		String token="Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";

		String payload=" {\r\n"

				+ "        \"status\": \"active\"\r\n"
				+ "    }";
Response response=given()
	.header("Authorization",token)
		//we can put the Content Type on Header as Postman
		.body(payload)
		.header("Content-Type","application/json")
		//Second way:
		//.contentType(ContentType.JSON)
		.when()
		.patch(url+7736750);

		response.prettyPrint();

		//Get status code => it returns integer
				int statusCode=response.getStatusCode();
				System.out.println(statusCode);

	}}
```



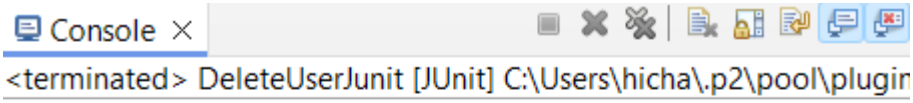
If we run delete methog again, we will get this : 404 because ressource not found



Delete Method Using @Test Annotation for user : 7736750

```
package gorest;

import static io.restassured.RestAssured.given;
import org.junit.Test;

import io.restassured.response.Response;

public class DeleteUserJunit {
	@Test
	public void deletejunit() {
		String url="https://gorest.co.in/public/v1/users/7736750";

Response response=given()
		.header("Authorization","Bearer 3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
		.when()
```

```
                .delete(url);
                response.prettyPrint();

                //Get status code => it returns integer
                            int statusCode=response.getStatusCode();
                            System.out.println(statusCode);


        }
    }
```



204

We got 204 Because it was deleted !

Home Work End to End

1. Create User with Post method Validate staus code is 201

Validate response conetent type is JSON Extract record ID from response

2. Validate user created with GET BYID Validate Status code is 200

3. Update user's name and Email with PUT/Patch method Validate Status code is 200

4. Validate user name and email updated with GET BYID Validate Status code is 200

5.Delete user

Validate Status code is 204

6.Validate user deleted with GET BYID

Validate Status code is 404

Valite message "Resource not found

```
package gorest;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
public class End2End {
String url="https://gorest.co.in/public/v1/users";
String token="Bearer
1b0a651f0ee6151ed5586b675cd871b4178f23f1ee6900f0fb7a34d787ac1855";
String postPayload="{\r\n"
+ " \"name\": \"Cohort12\",\r\n"
+ " \"email\": \"ssssre@gmail.com\",\r\n"
+ " \"gender\": \"male\",\r\n"
+ " \"status\": \"active\"\r\n"
```

```
+ "}";
String patchPayload="{\r\n"
+ " \"name\": \"YollStudents\",\r\n"
+ " \"email\": \"cohort12@yoll.com\"\r\n"
+ "}";
@Test
public void homework() {

// 1. Create User with Post method
Response postResponse=given()
.header("Authorization",token)
.contentType(ContentType.JSON)
.body(postPayload)
.when()
.post(url);
postResponse.prettyPrint();

// Validate status code is 201
//postResponse.then().statusCode(201);
assertEquals(201, postResponse.getStatusCode());

// Validate response type is JSON
postResponse.then().contentType(ContentType.JSON);

// Extract record ID from response
int id=postResponse.then().extract().path("data.id");
System.out.println(id);

// 2. Validate user created with GET ByID
Response getResp=given()
.header("Authorization",token)
.when()
.get(url+"/"+id);
getResp.prettyPrint();

// Validate Status code is 200
//getResp.then().statusCode(200);
assertEquals(200, getResp.getStatusCode());

// 3. Update user's name and Email with PUT/Patch method
Response patchResp=given()
.header("Authorization",token)
.contentType(ContentType.JSON)
.body(patchPayload)
.when()
.patch(url+"/"+id);
```

```
patchResp.prettyPrint();

// Validate Status code is 200
assertEquals(200, patchResp.getStatusCode());

// 4. Validate user name and email updated with GET ByID
String email=patchResp.then().extract().path("data.email");
String name=patchResp.then().extract().path("data.name");
assertEquals("cohort12@yoll.com", email);
assertEquals("YollStudents", name);

// Validate Status code is 200
assertEquals(200, patchResp.getStatusCode());

// 5.Delete user
Response deleteResp=given()
.header("Authorization",token)
.when()
.delete(url+"/"+id);

// Validate Status code is 204
assertEquals(204, deleteResp.getStatusCode());

// 6.Validate user deleted with GET ByID
Response getResp2=given()
.header("Authorization",token)
.when()
.get(url+"/"+id);

// Validate Status code is 404
assertEquals(404, getResp2.getStatusCode());

// Validate message "Resource not found
String errMessage=getResp2.then().extract().path("data.message");
System.out.println(errMessage);
assertEquals("Resource not found",errMessage );
}
}
```

Result:

An "end-to-end scenario" in the context of API testing with RestAssured typically involves testing the entire workflow or sequence of actions that a user might perform using the API. This goes beyond testing individual endpoints in isolation and includes testing the interactions between multiple endpoints to ensure the entire process works as expected.

Example End-to-End Scenario

Let's consider an example where we test the workflow of creating a user, retrieving the user details, updating the user, and then deleting the user.

Step-by-Step Breakdown

1. **Create a User**: Send a POST request to create a new user.

2. **Retrieve User Details**: Send a GET request to retrieve the details of the newly created user.
3. **Update User**: Send a PUT or PATCH request to update the user's details.
4. **Delete User**: Send a DELETE request to remove the user.
5. **Verify Deletion**: Send a GET request to ensure the user no longer exists.

Implementing End-to-End Scenario with RestAssured

Here is an example implementation in Java using RestAssured and JUnit:

Explanation

- **Base URL and Token**: The `baseUrl` and `token` are defined once to be reused in all requests.
- **Step 1: Create a User**: A POST request is sent to create a user. The response is checked to ensure the status code is `201` (Created). The `userId` is extracted from the response for use in subsequent steps.
- **Step 2: Retrieve User Details**: A GET request is sent to retrieve the details of the newly created user. The response is checked to ensure the status code is `200` (OK).
- **Step 3: Update User**: A PUT request is sent to update the user's name. The response is checked to ensure the status code is `200` (OK).
- **Step 4: Delete User**: A DELETE request is sent to remove the user. The response is checked to ensure the status code is `204` (No Content), indicating successful deletion.
- **Step 5: Verify Deletion**: A GET request is sent to ensure the user no longer exists. The response is checked to ensure the status code is `404` (Not Found).

This complete end-to-end test ensures that the entire workflow for creating, retrieving, updating, and deleting a user works as expected.

```
package gorest;

import org.junit.Test;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

import io.restassured.response.Response;

public class EndToEndTest {
```

```
    private String baseUrl = "https://gorest.co.in/public/v1";
    private String token = "Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7";

    @Test
    public void endToEndUserScenario() {

        // Step 1: Create a User
        String createUserPayload = "{\r\n"
                + "    \"name\": \"Cohort12\",\r\n"
                + "    \"email\": \"abcd123@gmail.com\",\r\n"
                + "    \"gender\": \"male\",\r\n"
                + "    \"status\": \"active\"\r\n"
                + "}";

        Response createResponse = given()
                .header("Authorization", token)
                .header("Content-Type", "application/json")
                .body(createUserPayload)
                .when()
                .post(baseUrl + "/users");

        createResponse.then().statusCode(201); // Check if user
creation is successful

        int userId = createResponse.jsonPath().getInt("data.id");

        // Step 2: Retrieve User Details
        Response getResponse = given()
                .header("Authorization", token)
                .when()
                .get(baseUrl + "/users/" + userId);

        getResponse.then().statusCode(200); // Check if user
retrieval is successful

        // Step 3: Update User
        String updateUserPayload = "{\r\n"
                + "    \"name\": \"Updated Cohort12\"\r\n"
                + "}";

        Response updateResponse = given()
                .header("Authorization", token)
                .header("Content-Type", "application/json")
                .body(updateUserPayload)
                .when()
                .put(baseUrl + "/users/" + userId);
```

```
        updateResponse.then().statusCode(200); // Check if user
update is successful

        // Step 4: Delete User
        Response deleteResponse = given()
                .header("Authorization", token)
                .when()
                .delete(baseUrl + "/users/" + userId);

        deleteResponse.then().statusCode(204); // Check if user
deletion is successful

        // Step 5: Verify Deletion
        Response verifyDeletionResponse = given()
                .header("Authorization", token)
                .when()
                .get(baseUrl + "/users/" + userId);

        verifyDeletionResponse.then().statusCode(404); // Check if
user is really deleted
    }
}
```

OpenWeatherMap TASK : Built-in API request by city name

Click on: **Built-in API request by city name**

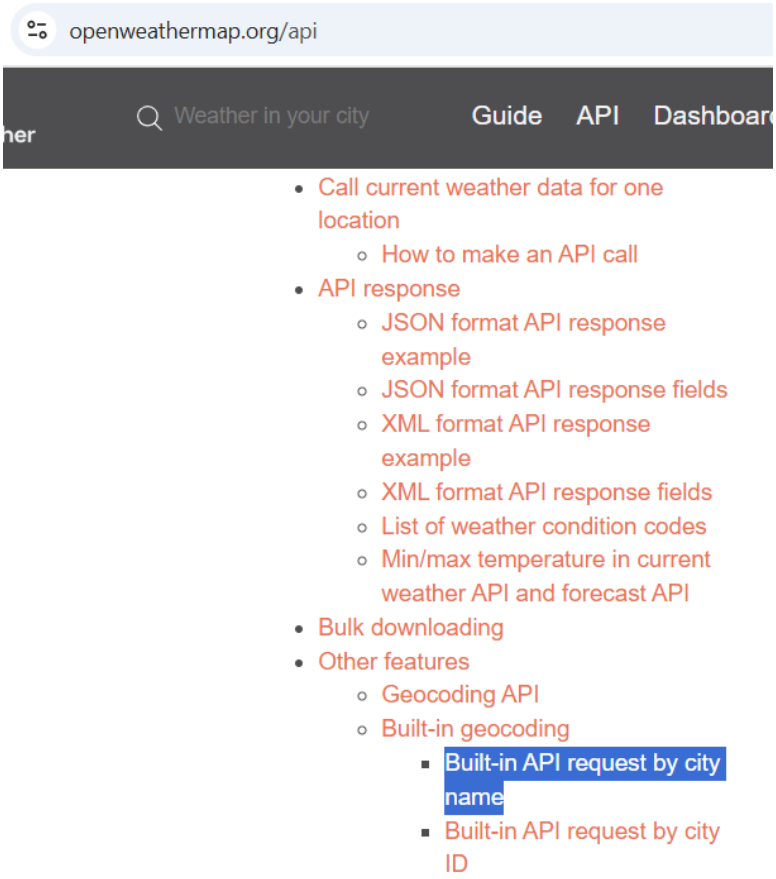
Get the weather of Paris and Rabat

Params query is : q

Open the website: **Weather API - OpenWeatherMap**

URL : **https://api.openweathermap.org/data/2.5/weather**

API KEY : appid : 926157617b49a877ea7e848f022353a9



You can find the url here:

Built-in API request by city name

You can call by city name or city name, state code and country code. Please note that searching by states available only for the USA locations.

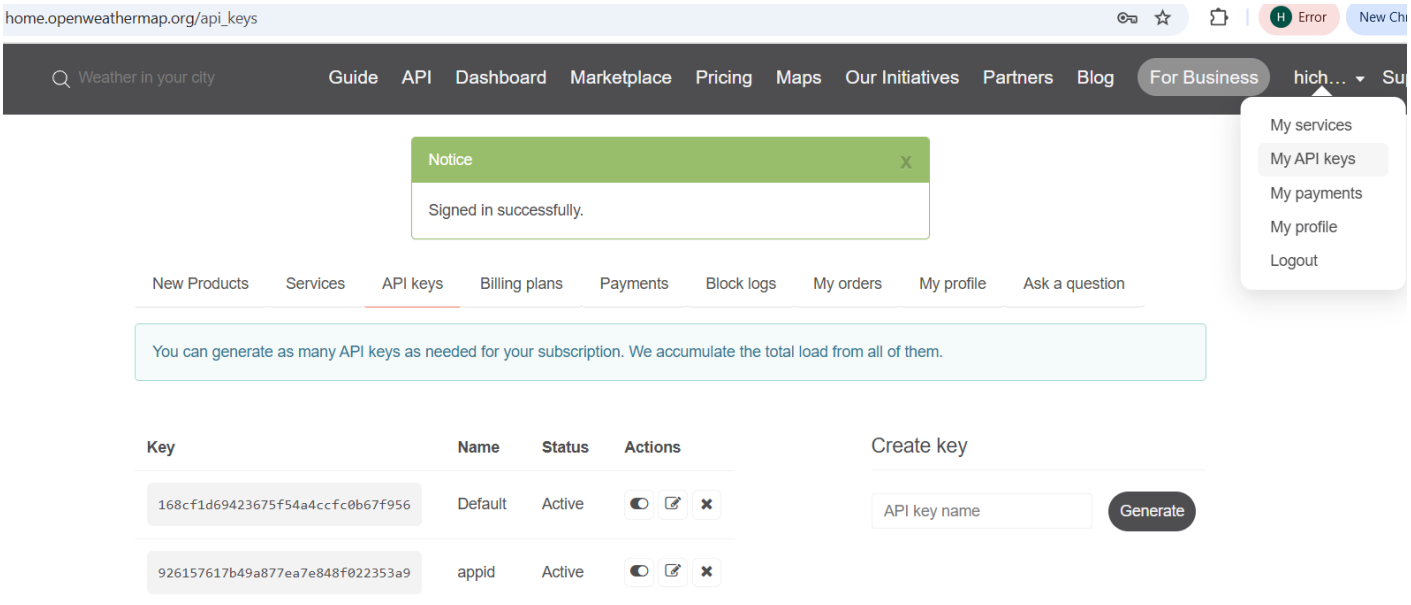
API call

```
https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
```

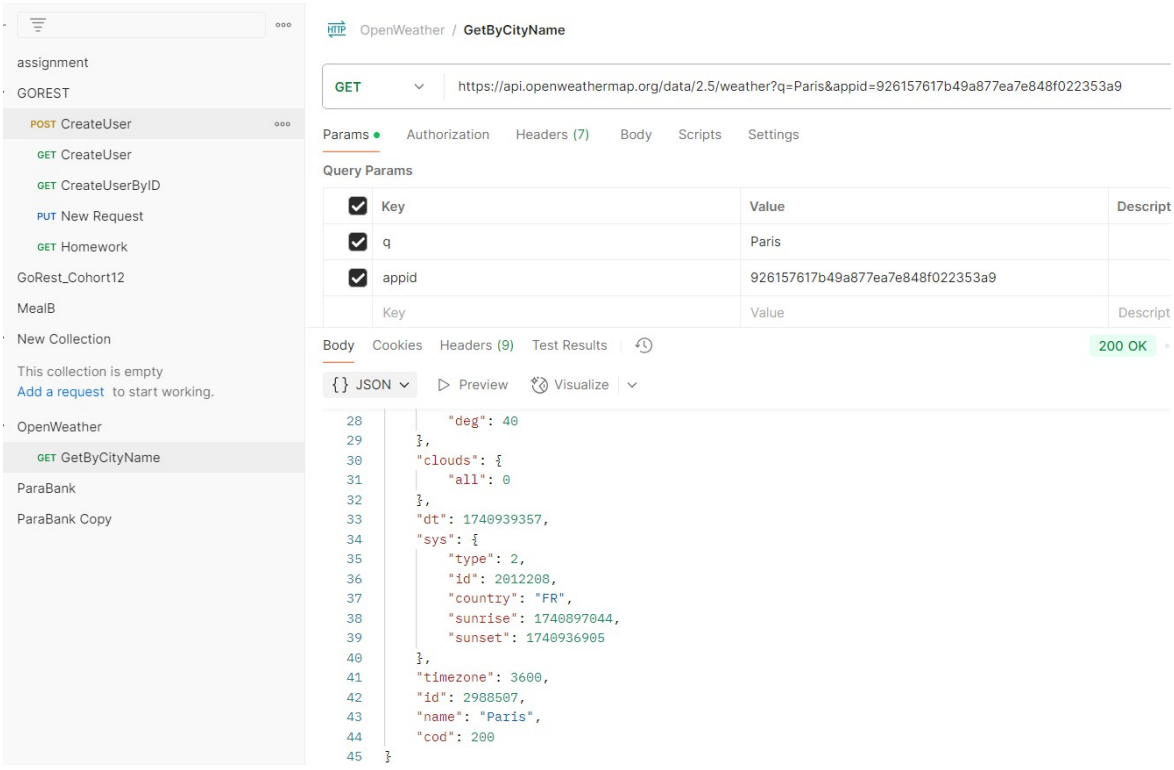
```
https://api.openweathermap.org/data/2.5/weather?q={city name},{country code}&appid={API key}
```

```
https://api.openweathermap.org/data/2.5/weather?q={city name},{state code},{country code}&appid={API key}
```

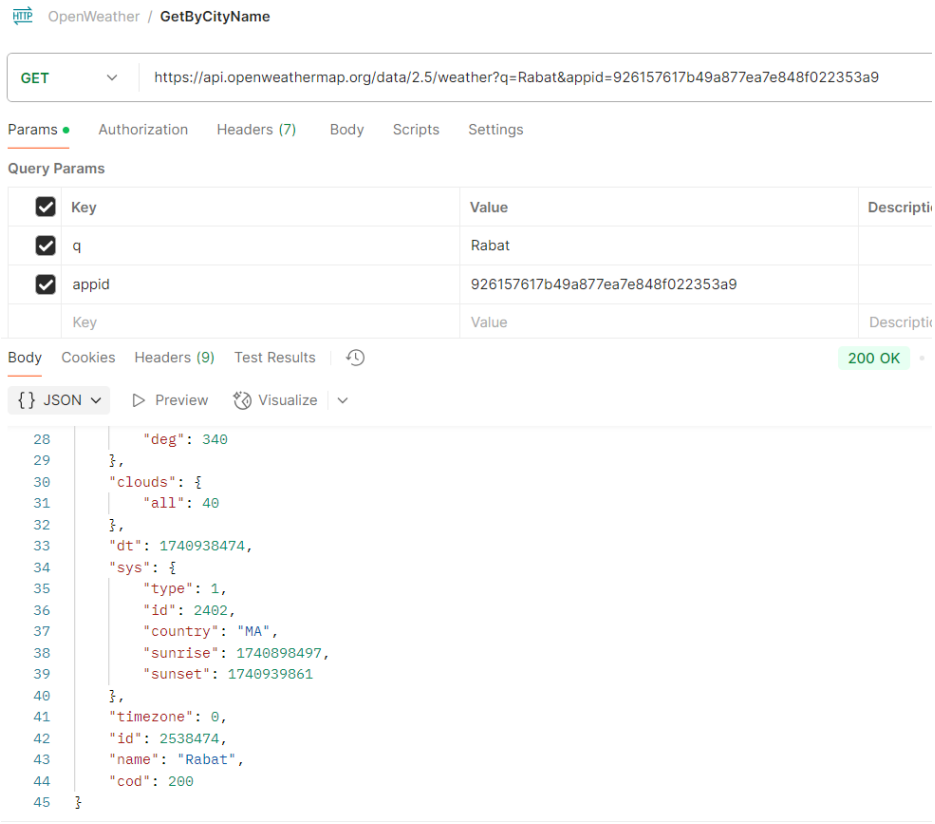
You can find the key here:



Create a Get method in Postman: Paris



Rabat:

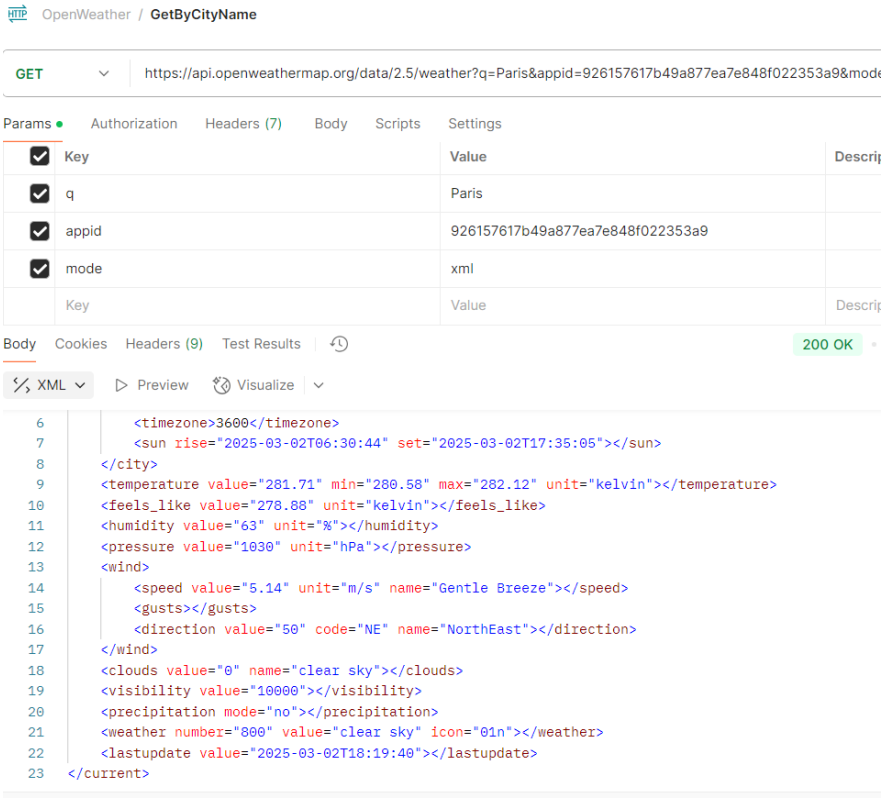


Parameters

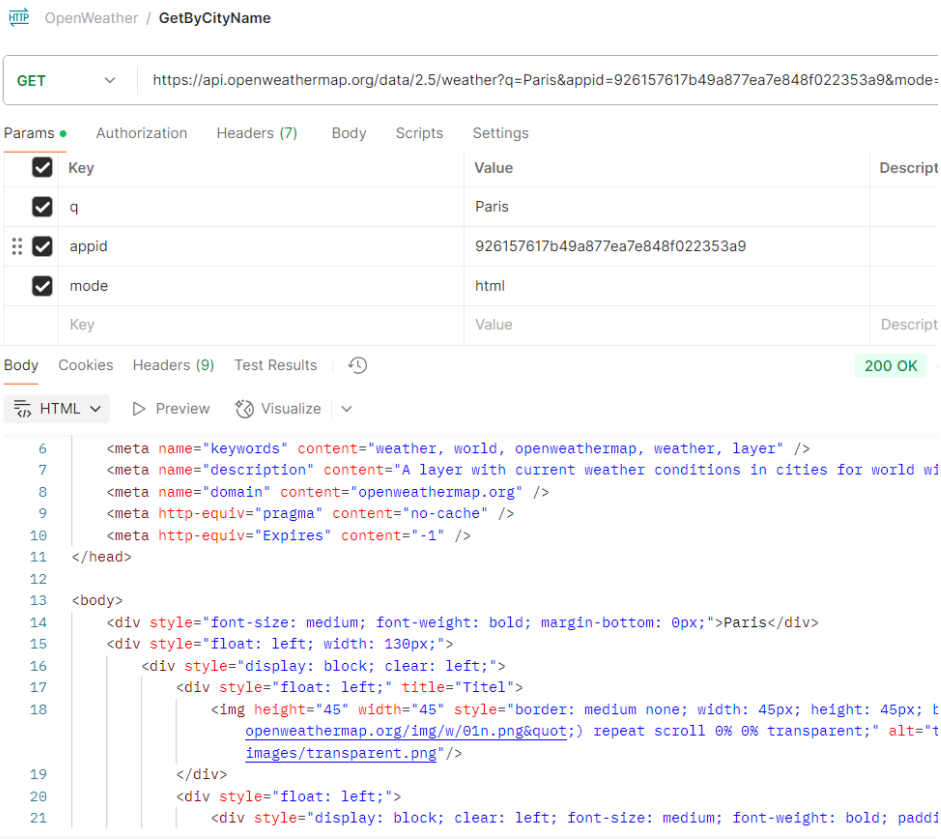
`q`	required	City name, state code and country code divided by comma, Please refer to ISO 3166 for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.
`appid`	required	Your unique API key (you can always find it on your account page under the "API key" tab)
`mode`	optional	Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. Learn more
`units`	optional	Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. Learn more
`lang`	optional	You can use this parameter to get the output in your language. Learn more

MODE : Write mode on Query Params to get the data on different Formats: XML, HTML. JSON is a default format.

XML Format:

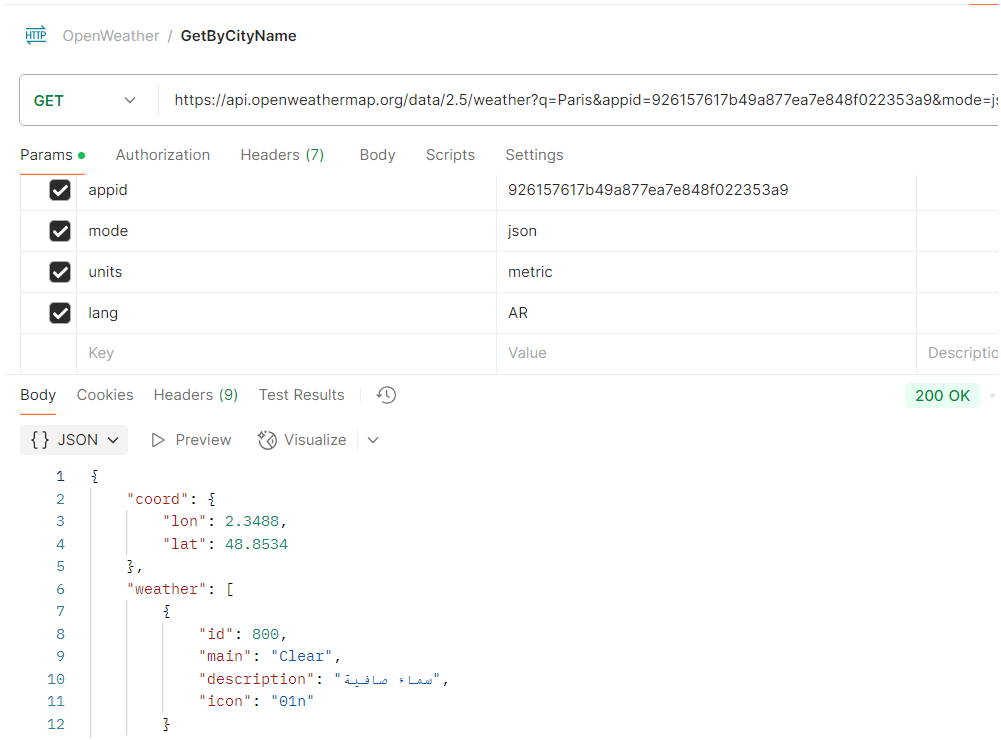


HTML Format:



UNITS : Write UNITS on Query Params to get the data on different units of measurments: metric, standard, imperial is a default format.

LANG: Write lang on Query Params to get the data on different Languages: English, Arabic.



TASK :Built-in API request by city ID

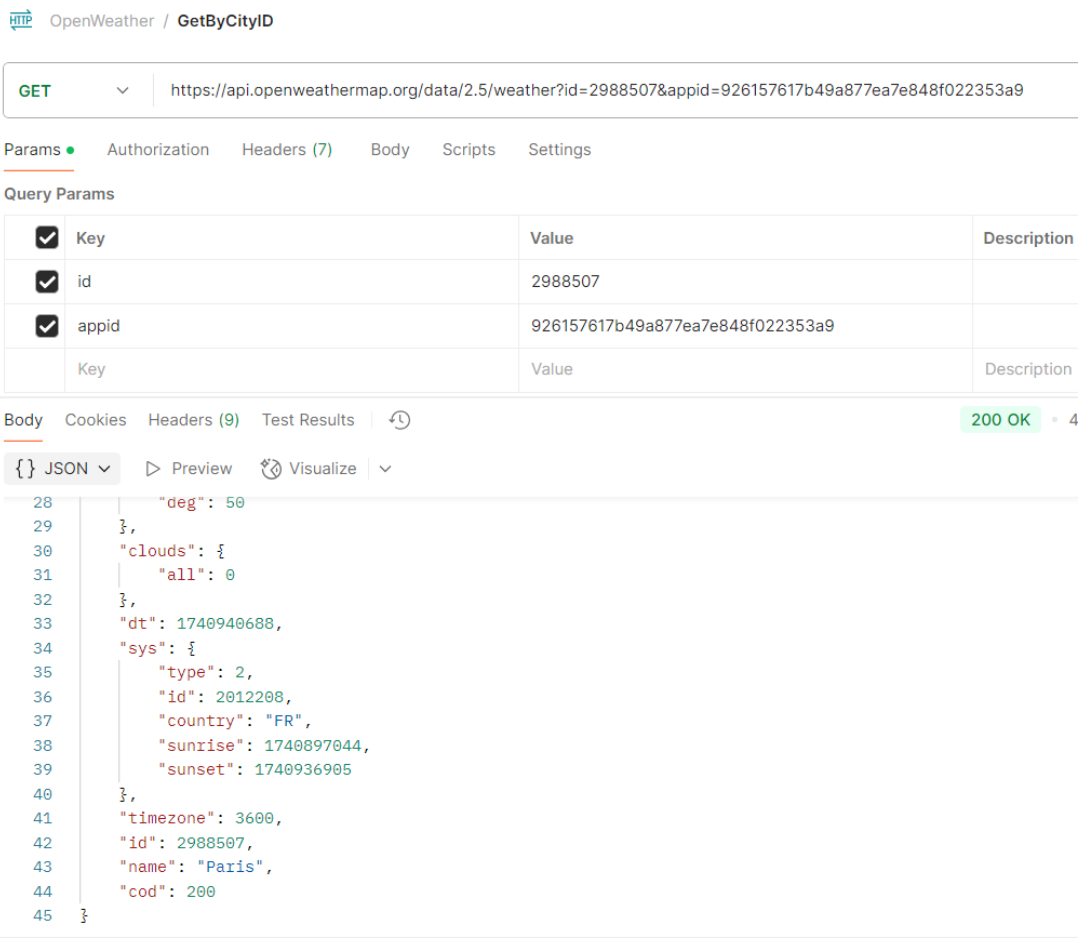
You can make an API call by city ID. Each city has an ID

API call

```
https://api.openweathermap.org/data/2.5/weather?id={city id}&appid={API key}
```

Parameters

`id`	required	City ID. List of city ID 'city.list.json.gz' can be downloaded here.
`appid`	required	Your unique API key (you can always find it on your account page under the "API key" tab)
`mode`	optional	Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. Learn more
`units`	optional	Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. Learn more
`lang`	optional	You can use this parameter to get the output in your language. Learn more



Task : Built-in API request by ZIP code

API call

```
https://api.openweathermap.org/data/2.5/weather?zip={zip code},{country code}&appid={API key}
```

Parameters

`zip`	required	Zip code
`appid`	required	Your unique API key (you can always find it on your account page under the "API key" tab)
`mode`	optional	Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. Learn more
`units`	optional	Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. Learn more
`lang`	optional	You can use this parameter to get the output in your language. Learn more

OpenWeather / **GetByCityID**

GET https://api.openweathermap.org/data/2.5/weather?zip=27606&appid=926157617b49a877ea7e848f022353a9

Params • Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Descriptio
zip	27606	
appid	926157617b49a877ea7e848f022353a9	
Key	Value	Descriptio

Body Cookies Headers (9) Test Results 200 OK

{ } JSON Preview Visualize


```
        "deg": 40
    },
    "clouds": {
        "all": 20
    },
    "dt": 1740941047,
    "sys": {
        "type": 2,
        "id": 2074833,
        "country": "US",
        "sunrise": 1740915805,
        "sunset": 1740957047
    },
    "timezone": -18000,
    "id": 0,
    "name": "Raleigh",
    "cod": 200
}
```

RETRIEVE AN ELEMENT VALUE FROM JSON RESPONSE

We can retrieve an element value from json response in 2 ways:

1→ we can use **extract().path()** methods and need to provide path of that element.

Example: the path of "**total**" in this json data is: **meta.pagination.total**

```
Response response=given().when().get(url);

int total=response.then().extract().path("meta.pagination.total");
```

2→ we can use **JsonPath.**

Add dependency:

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>2.7.0</version>
</dependency>
```

Use **JsonPath.read()** retrieve values from json data.

```
JsonPath.read(jsonData, "$.yourJsonPath");
```

JSON PATH

Use this website:

JSONPath Online Evaluator

JSONPath is a query language for JSON, similar to XPath for XML. It allows you to select and extract data from a JSON document. You use a **JSONPath expression** to traverse the path to an element in the JSON structure. You start at the root node or element, represented by *$*, and reach the required element in the JSON structure to extract data from it. You can use either the dot-notation or the bracket-notation to form the expressions.

xpression	Description
`$`	The root node or element.
`*`	Wildcard. Selects all elements in an object or array. For example, `address.*` returns all properties of the `address` object, and `book[*]` returns all items of the `book` array.
`..<name>`	A recursive scan. Selects and returns a list of all the properties of the element that matches `<name>`. The element referenced must be a JSON object. For example, consider the following JSON structure: The expression `$..data` returns a list of all the properties of the object, `data`.
`.<name>`	The dot-notated child element of the current node. For example, `$.event`
`['<name>', '<name>',...]`	The bracket-notated child or children of the current node. For example, the expression `$['event']['type', 'data']` returns all the properties of the child elements, `type` and `data`, of the node, `event`. **Note**: This notation is useful if the property name contains special characters, such as spaces, or begins with a character other than *A..Za..z_*. For example, `$['personal info']`
`[n]`	The *n*-th element of an array. Indices start from 0.

xpression	Description
`[<index1>, <index2>,...]`	The array elements specified by the given indices. The result is a list of elements.
`[start:end]` `[start:]`	The array elements from the *start* index up to, but not including, the *end* index. If *end* is omitted, selects all elements from *start* until the end of the array. The result is a list of elements.
`[:n]`	The first *n* elements of the array. The result is a list of elements.
`[-n:]`	The last *n* elements of the array. The result is a list of elements.
`[?(<expression>)]`	A **filter expression**. The expression must return a boolean value. The result is a list of all the elements in an object or array that match the specified filter.
`@`	The current node or element being processed in the filter expression.

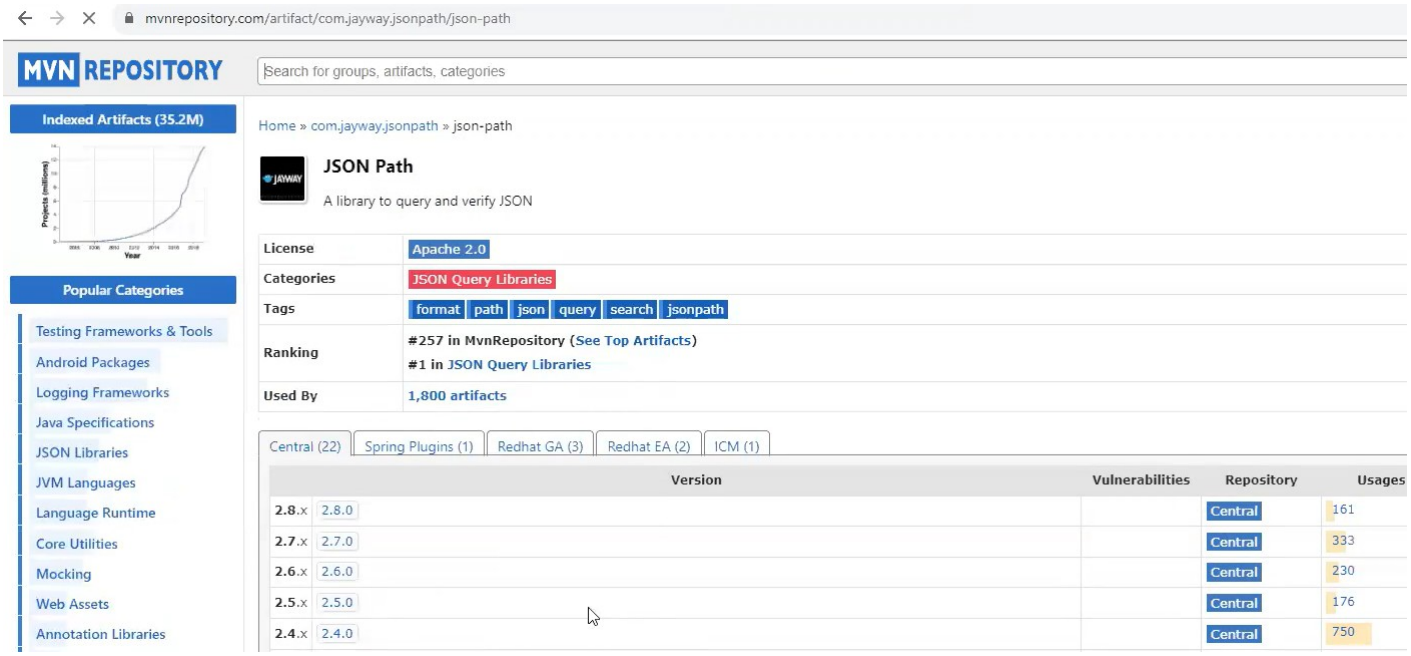
Operator	Description
`==`	Equal to. For example, `[?(@.color == 'blue')]` matches elements whose color is *blue*. **Note**: In the number to string comparison, 1 is not equal to '1'.
`!=`	Not equal to. For example, `[?(@.color != 'red')]` matches elements whose color is *not red*.
`<`	Less than. For example, `[?(@.days < 30)]` matches elements in which the number of days is less than *30*.
`<=`	Less than or equal to. For example, `[?(@.days <= 30)]` matches elements in which the number of days is less than or equal to *30*.
`>`	Greater than. For example, `[?(@.days > 30)]` matches elements in which the number of days is greater than *30*.
`>=`	Greater than. For example, `[?(@.days >= 30)]` matches elements in which the number of days is greater than or equal to *30*.
`=~`	Matches a regular expression. For example, `[?(@.firstName =~ /vi.*?/i)]` matches elements whose firstName starts with *vi* (case-insensitive).
`!`	Negates a filter. For example, `[?(!@.isbn)]` matches elements that do not have the isbn property.
`&&`	Logical AND. Used to combine multiple filter expressions where all the conditions must be met. For example, `[?(@.isbn && @.price < 10)]` matches elements that have an ISBN, and whose price is less than *10*.
`||`	Logical OR. Used to combine multiple filter expressions where any of the

Operator	Description
	conditions may be met. For example, `[?(@.category == 'fiction' || @.price < 10)]` matches elements whose category is *fiction* or whose price is less than *10.*
in	Checks if the value on the left exists in the set on the right. For example, `[?(@.size in ['S', 'M'])]` matches elements whose size is either '*S*' or '*M*'. **Note**: String comparison is case-sensitive.
nin	Checks if the value on the left is not in the set on the right. For example, `[?(@.size nin ['S', 'M'])]` matches elements whose size is neither '*S*' nor '*M*'. **Note**: String comparison is case-sensitive.
size	Checks if the size of the array or the string matches the specified value. For example, `[?(@.name size 12)]` matches all elements whose name length is *12*, including spaces.
empty true	Matches an empty array or string. For example, `[?(@.isbn empty true)]` matches all elements whose isbn property has an empty value.
empty false	Matches a non-empty array or string. For example, `[?(@.isbn empty false)]` matches all elements whose isbn property has a non-empty value.

Read and Write on a file using JSON PATH Dependency

```
<!--
https://mvnrepository.com/artifact/com.jayway.jsonpath
/json-path -->
        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
            <version>2.7.0</version>
        </dependency>
```

Get `jsonpath` from : https://mvnrepository.com



The dependency **com.jayway.jsonpath:json-path** is used to work with **JSONPath**, a query language for JSON documents. JSONPath allows you to navigate, extract, and manipulate data from JSON structures, similar to how XPath works for XML. This library is particularly useful in API testing, data validation, and JSON data processing.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3
    <modelVersion>4.0.0</modelVersion>
    <groupId>API</groupId>
    <artifactId>API</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>5.3.0</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>

        <dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>2.7.0</version>
</dependency>
```

Save and update the Project

Purpose of json-path Dependency

1. **JSON Data Extraction**:

    - Extract specific values or subsets of data from a JSON document using JSONPath expressions.
2. **Data Validation**:
    - Validate JSON responses in API testing by checking if specific fields or values exist and match expected results.
3. **Dynamic Data Handling**:
    - Handle dynamic JSON structures where the data you need is nested or located at different levels.
4. **Simplified JSON Processing**:
    - Avoid manually parsing JSON using libraries like org.json or Jackson when you only need to extract specific parts of the JSON.

Key Features of json-path

- **JSONPath Expressions**:
    - Supports JSONPath syntax to query JSON documents (e.g., $.store.book[0].title).
- **Filtering**:
    - Allows filtering JSON arrays based on conditions (e.g., $[?(@.price < 10)]).
- **Integration**:
    - Works seamlessly with other libraries like RestAssured for API testing.
- **Flexibility**:
    - Can handle both simple and complex JSON structures.

Example Use Cases

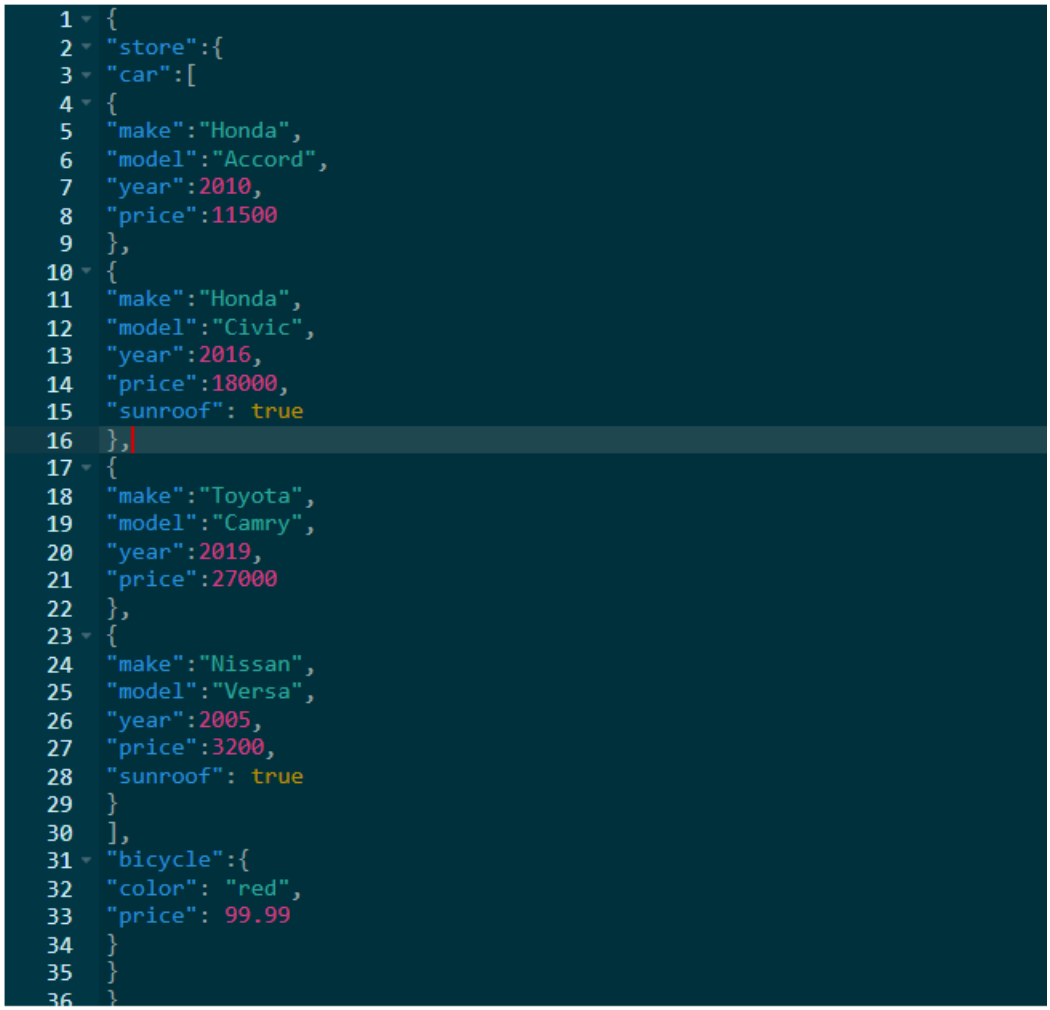
1. **Extracting Data from JSON**:
    - Extract specific fields or values from a JSON response.
2. **Validating API Responses**:
    - Check if a JSON response contains specific data or matches expected values.
3. **Dynamic Data Processing**:
    - Handle JSON data where the structure or location of data is not fixed.

How to Use json-path

1. Add the Dependency

Include the json-path dependency in your pom.xml

Inputs

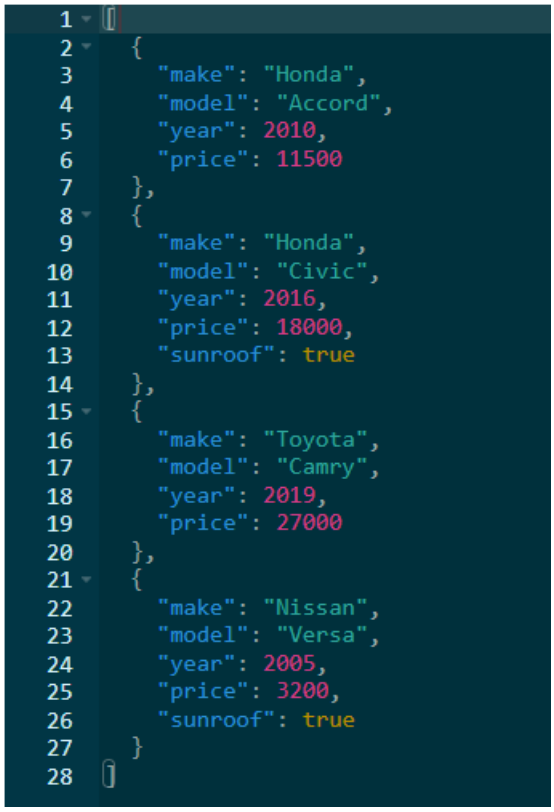


{
"store":{
"car":[
{
"make":"Honda",
"model":"Accord",
"year":2010,
"price":11500
},
{
"make":"Honda",
"model":"Civic",
"year":2016,
"price":18000,
"sunroof": true
},
{
"make":"Toyota",
"model":"Camry",
"year":2019,
"price":27000
},
{
"make":"Nissan",
"model":"Versa",
"year":2005,
"price":3200,
"sunroof": true
}
],
"bicycle":{
"color": "red",
"price": 99.99
}
}
}

"model":"Civic",
"year":2016,
"price":18000,
"sunroof": true
},
{
"make":"Toyota",
"model":"Camry",
"year":2019,
"price":27000
},
{
"make":"Nissan",
"model":"Versa",
"year":2005,
"price":3200,
"sunroof": true
}
],
"bicycle":{
"color": "red",
"price": 99.99
}
}
}

$.store.car[*] --gets all objects in car array

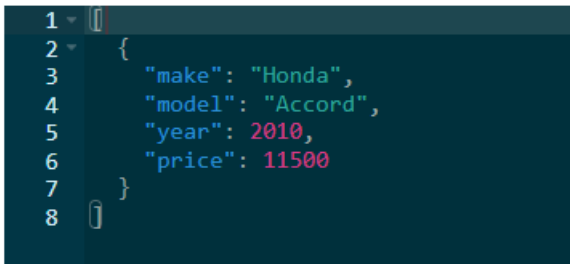
Evaluation Results




```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  },
  {
    "make": "Toyota",
    "model": "Camry",
    "year": 2019,
    "price": 27000
  },
  {
    "make": "Nissan",
    "model": "Versa",
    "year": 2005,
    "price": 3200,
    "sunroof": true
  }
]
```


$.store.car[0] --gets 1st object in car array

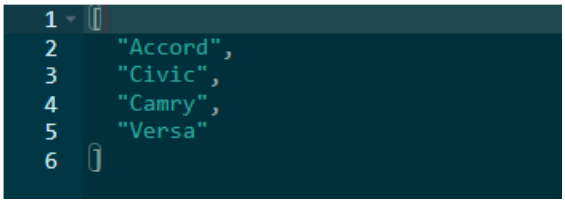
Evaluation Results




```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  }
]
```


$.store.car[*].model --gets model of all cars

Evaluation Results




```
[
  "Accord",
  "Civic",
  "Camry",
  "Versa"
]
```


$.store.car[0].model --gets model of 1st car

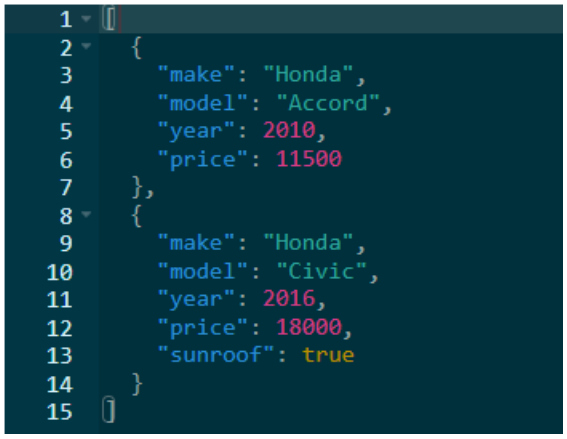
Evaluation Results

$.store.car[0,1,2].price -- gets price of 1st, 2nd, 3rd cars

Evaluation Results

$.store.car[:2] -- gets first 2 object in car array

Evaluation Results

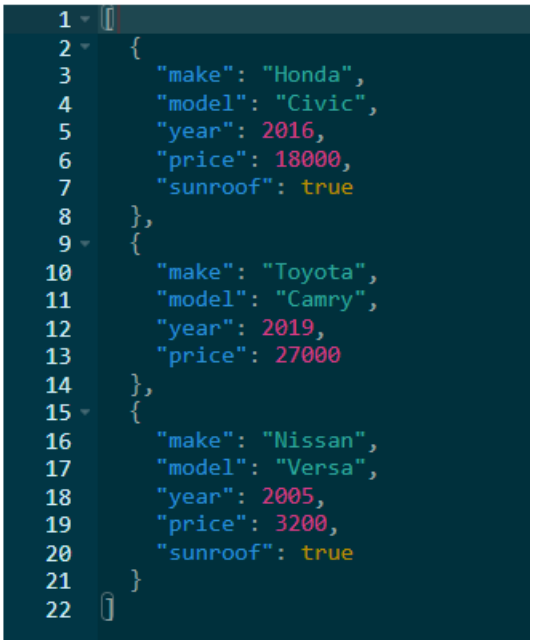


$.store.car[-2:] -- gets last 2 object in car array

Evaluation Results

$.store.car[1:] --gets all objects starting from index 1

Evaluation Results




```
[
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  },
  {
    "make": "Toyota",
    "model": "Camry",
    "year": 2019,
    "price": 27000
  },
  {
    "make": "Nissan",
    "model": "Versa",
    "year": 2005,
    "price": 3200,
    "sunroof": true
  }
]
```


$.store.car[?(@.make=="Honda")] gets car which model is Honda

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  }
]
```


$.store.car[?(@.price<15000)] gets car which price<15000

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Nissan",
    "model": "Versa",
    "year": 2005,
    "price": 3200,
    "sunroof": true
  }
]
```

$.store.car[?(@.sunroof)] gets car with sunroof

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  },
  {
    "make": "Nissan",
    "model": "Versa",
    "year": 2005,
    "price": 3200,
    "sunroof": true
  }
]
```

$.store.car[?(@.make=="Honda" & @.year>2015)] gets honda which year>2015

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  }
]
```

$.store.car[?(@.make=="Honda" | @.year>2015)] gets all hondas and other cars which year>2015

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Nissan",
    "model": "Versa",
    "year": 2005,
    "price": 3200,
    "sunroof": true
  }
]
```

$.store.car[? (@.year>=2010 & @.year<2017)] get cars which year between 2010 and less than 2017

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  }
]
```

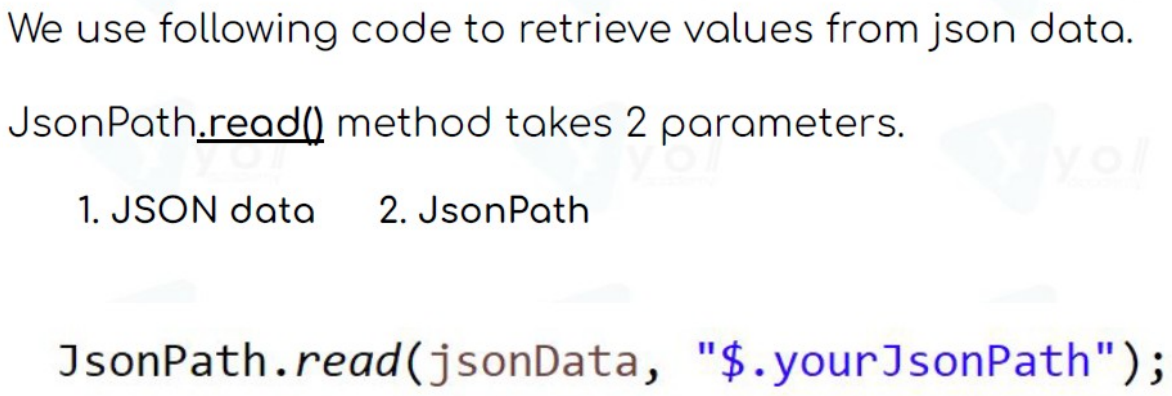
$.store.car[?(@.make == "Honda" & @.price>10000)]–get Honda where price>10000

Evaluation Results

```
[
  {
    "make": "Honda",
    "model": "Accord",
    "year": 2010,
    "price": 11500
  },
  {
    "make": "Honda",
    "model": "Civic",
    "year": 2016,
    "price": 18000,
    "sunroof": true
  }
]
```

Json Path Practice:

This Java code demonstrates how to use RestAssured and JsonPath to test a REST API. It specifically retrieves a list of users filtered by gender (male) from the GoRest API and verifies that all users in the response are indeed male.


We use following code to retrieve values from json data.
JsonPath.read() method takes 2 parameters.
1. JSON data
2. JsonPath
JsonPath.read(jsonData, "$.yourJsonPath");

Retrieve the male:

```
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import java.util.List;
import io.restassured.response.Response;

public class JsonPathPractice {
	String url = "https://gorest.co.in/public/v1/users";
	// Filter gorest male user and
	// validate all user in response are male
```

```
	@Test
	public void testGenderFilter() {
		Response resp = given().param("gender", "male").when().get(url);
		resp.prettyPrint();
		List<String> genderList = JsonPath.read(resp.asString(),
"$.data[*].gender");

		for (String eachGender : genderList) {
			System.out.println(eachGender);
			assertEquals("male", eachGender);
		}

	}
}
```

Explaination:

```
package gorest;

//Import necessary JUnit libraries for testing
import org.junit.Test;

//Import JsonPath library for extracting data from JSON responses
import com.jayway.jsonpath.JsonPath;

//Import RestAssured methods for API testing
import static io.restassured.RestAssured.given;

//Import JUnit assertion to validate test cases
import static org.junit.Assert.assertEquals;

//Import Java List class to store extracted JSON values
import java.util.List;

//Import RestAssured Response class to handle API responses
import io.restassured.response.Response;

//Define the test class
public class JsonPathPractice {

	// Define the base URL for the GoRest API
	String url = "https://gorest.co.in/public/v1/users";

	// This test filters male users from the API response
	// and validates that all returned users have gender = "male"
	@Test
	public void testGenderFilter() {

		// Send a GET request with a query parameter "gender=male"
		Response resp = given() // Start RestAssured request
```

```
                    .param("gender", "male") // Add query parameter:
gender=male
                    .when() // Specify the HTTP method
                    .get(url); // Make a GET request to the API URL

          // Print the response body in a readable format
          resp.prettyPrint();

          // Extract the list of gender values from the JSON
response using JSONPath

//JsonPath.read("jsondata","jsonpath");
          List<String> genderList = JsonPath.read(resp.asString(),
"$.data[*].gender");

          // Loop through each extracted gender value and validate
it is "male"
          for (String eachGender : genderList) {
               System.out.println(eachGender); // Print each gender
from the response
               assertEquals("male", eachGender); // Assert that each
gender is "male"
          }
     }
}
```

Result:


```
{
    "meta": {
        "pagination": {
            "total": 1419,
            "pages": 142,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    },
    "data": [
        {
            "id": 6958844,
            "name": "Bhadran Verma Jr.",
            "email": "bhadran_jr_verma@sawayn-yost.example",
            "gender": "male",
            "status": "inactive"
        },
        {
```

```
    "id": 6958843,
    "name": "Marut Chopra",
    "email": "marut_chopra@hansen.example",
    "gender": "male",
    "status": "inactive"
},
{
    "id": 6958842,
    "name": "Advaya Guha",
    "email": "guha_advaya@torphy-zieme.example",
    "gender": "male",
    "status": "inactive"
},
{
    "id": 6958841,
    "name": "Bhagvan Kaul",
    "email": "kaul_bhagvan@streich-stokes.test",
    "gender": "male",
    "status": "active"
},
{
    "id": 6958838,
    "name": "The Hon. Garud Dhawan",
    "email": "garud_hon_the_dhawan@fahey.test",
    "gender": "male",
    "status": "active"
},
{
    "id": 6958834,
    "name": "Shreya Kocchar",
    "email": "shreya_kocchar@orn-yost.example",
    "gender": "male",
    "status": "active"
},
{
    "id": 6958833,
    "name": "Mrs. Chandraketu Sharma",
    "email": "mrs_sharma_chandraketu@ullrich.test",
    "gender": "male",
    "status": "inactive"
},
{
    "id": 6958832,
    "name": "Radha Bharadwaj",
    "email": "radha_bharadwaj@schulist-olson.test",
    "gender": "male",
    "status": "inactive"
},
{
    "id": 6958102,
    "name": "Siddarth Shukla",
    "email": "shukla_siddarth@hintz.test",
    "gender": "male",
    "status": "active"
},
{
    "id": 6958096,
    "name": "Akroor Ahuja",
    "email": "ahuja_akroor@feeney-corkery.test",
    "gender": "male",
    "status": "inactive"
```

```
        }
    ]
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
male
male
male
male
male
male
male
male
male
male
```

Retrieving all emails:

```java
package gorest;
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import java.util.List;
import io.restassured.response.Response;

public class UpdateUserPatchJunit {
     String url = "https://gorest.co.in/public/v1/users";
```

```
    @Test
    public void emailFilter() {
        Response resp = given().when().get(url);
        resp.prettyPrint();
        List<String> emails = JsonPath.read(resp.asString(),
"$.data[*].email");

        for (String email : emails) {
            System.out.println(email);
        }

    }
}
```

```
khatri_lakshmi@kilback.example
vasudeva_desai@vonrueden.test
devar_anuraag@jerde-abbott.example
prof_girija_bhat@mcglynn.test
menon_kanishka@price-paucek.example
verma_anala@beer.example
trivedi_brahma@fahey.example
mishra_dhanesh@mcdermott.test
suman_marar@anderson.test
abbott_jagadish@breitenberg.test
```

Reading Json file:

Use String, int, list<String> for different Varibles:

Retrieved values can be assigned to proper variables:

```
String name=    JsonPath.read(jsonData, "$.meta.name");

int pageNumber=  JsonPath.read(jsonData, "$.meta.pageNumber");

List<String>genderList= JsonPath.read(jsonData, "$.data[*].gender");
```

HOMEWORK ASSIGNMENT FOR INTERVIEW

1. Submit Get request to gorest API with following params:

name=bal

2. You will get 36 users in 4 pages.

(Counts may change depending on other people add/delete data)

3. Get value of "total" and "pages" elements

4. Submit GET request for each page and (use loop)

5. Retrieve all names from each response and add into a list

6. Validate list size is equals to value of "total"

(If you have total 36 users, there should be 36 names in list)

7. Validate all names in list contains "bal"

Notes and Tips:

Use value of "pages" in loop to control how many times it will GET request

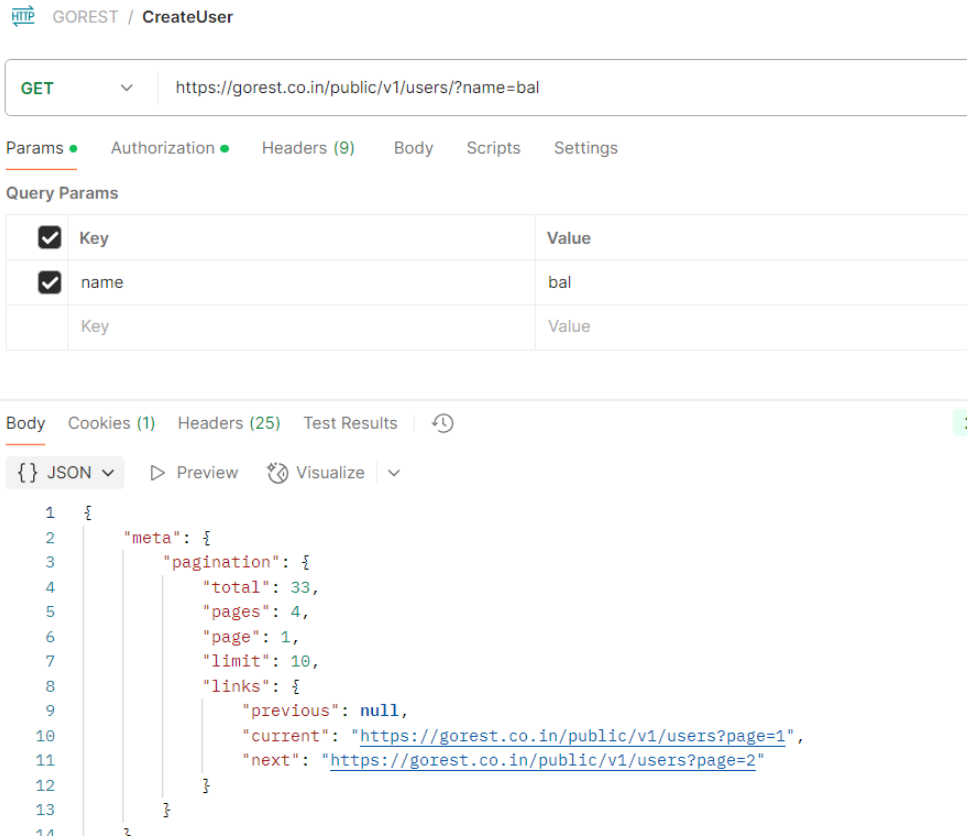
Use "page" param to to get names in different pages

Make your code dynamic so if more users with "bal" substring added/deleted, your code still works.

++++++++

We have 4 pages that contain 33 names by using the params `name contains "bal"`**:**

The first 3 pages contains 30 names, the fourth page contains 3 names.




```java
package homework;

import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import java.util.ArrayList;
import java.util.List;
import io.restassured.response.Response;

public class GenderParamValidation {
	List<String> namesList = new ArrayList<>();

	@Test
	public void testGenderParamFilter() {

// 1. Submit Get request to gorest API with following params:name=bal
		Response resp = given()
					.param("name", "bal")
				.when()
				.get("https://gorest.co.in/public/v1/users");
		resp.prettyPrint();

// 2. Get value of "total" and "pages" elements
		int total = JsonPath.read(resp.asString(), "$.meta.pagination.total");
		int pages = JsonPath.read(resp.asString(), "$.meta.pagination.pages");

		System.out.println("total: " + total);
		System.out.println("pages: " + pages);

// 3. Submit GET request for each page and (use loop)
```

```
            for (int i = 1; i <= pages; i++) {
                  Response response = given()
                                    .param("name", "bal")
                                    .param("page", i).when()
                                    .get("https://gorest.co.in/public/v1/users");
                  response.then().statusCode(200);

// 4. Retrieve all names from each response and add into a list
                  List<String> names = JsonPath.read(response.asString(),
"$.data[*].name");

// add 10 names from each page to namesList array;
                  namesList.addAll(names);

                  System.out.println(namesList.size());
            }
// 5. Validate list size is equals to value of "total"
// (If you have total 33 users, there should be 33 names in list )
            assertEquals(total, namesList.size());

// 6. Validate all names in list contains "bal"
            for (String eachName : namesList) {

//print name
                  System.out.println(eachName);

// validate each name contain "bal" substring
                  assertTrue(eachName.toLowerCase().contains("bal"));
            }
      }
}
```

RESULT:


```
{
    "meta": {
        "pagination": {
            "total": 34,
            "pages": 4,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    },
    "data": [
        {
            "id": 6972317,
            "name": "Prof. Balgopal Abbott",
            "email": "balgopal_abbott_prof@oconnell.example",
            "gender": "female",
            "status": "inactive"
        },
        {
            "id": 6945435,
```

```
    "name": "Gov. Bala Chopra",
    "email": "gov_chopra_bala@reichel-mayer.test",
    "gender": "male",
    "status": "active"
},
{
    "id": 6945417,
    "name": "Amb. Balachandra Mehra",
    "email": "balachandra_mehra_amb@olson.example",
    "gender": "female",
    "status": "inactive"
},
{
    "id": 6944466,
    "name": "Balagovind Jha",
    "email": "balagovind_jha@feil.example",
    "gender": "male",
    "status": "active"
},
{
    "id": 6944461,
    "name": "Balgopal Patel",
    "email": "patel_balgopal@koch.example",
    "gender": "male",
    "status": "active"
},
{
    "id": 6942866,
    "name": "Bala Acharya",
    "email": "acharya_bala@bernier.test",
    "gender": "male",
    "status": "active"
},
{
    "id": 6942435,
    "name": "Balachandra Varman",
    "email": "balachandra_varman@block-franecki.example",
    "gender": "female",
    "status": "inactive"
},
{
    "id": 6942421,
    "name": "Balachandra Chattopadhyay",
    "email": "balachandra_chattopadhyay@zulauf-yost.test",
    "gender": "female",
    "status": "active"
},
{
    "id": 6942340,
    "name": "Bala Shah",
    "email": "bala_shah@schinner-gutkowski.test",
    "gender": "female",
    "status": "inactive"
},
{
    "id": 6942048,
    "name": "Balamani Iyer CPA",
    "email": "balamani_iyer_cpa@morissette.example",
    "gender": "male",
    "status": "active"
}
```

```
    ]
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
total: 34
pages: 4
10
20
30
34
Prof. Balgopal Abbott
Gov. Bala Chopra
Amb. Balachandra Mehra
Balagovind Jha
Balgopal Patel
Bala Acharya
Balachandra Varman
Balachandra Chattopadhyay
Bala Shah
Balamani Iyer CPA
Balagopal Dubashi
Balagovind Saini
Balgopal Mehra PhD
Balgopal Menon Ret.
Balagovind Banerjee MD
Balachandra Panicker
Baladitya Varma
Rev. Balamani Malik
The Hon. Balgopal Gandhi
Balachandra Saini I
Balagovind Namboothiri
Balaaditya Sinha
Balaaditya Verma
Balaaditya Iyengar
Balgopal Ahluwalia
Balachandra Jain
Balaaditya Namboothiri LLD
Balachandra Sinha
Balagopal Marar
Bala Gill CPA
Prof. Baladitya Nair
Bala Ahluwalia
Balamani Nambeesan
Balachandra Kaul
```

Read & write external files

Data creating for Request

There are two ways of passing request body to PUT or POST methods:

1. Keep request data in an external file and read that content to String then pass it to method

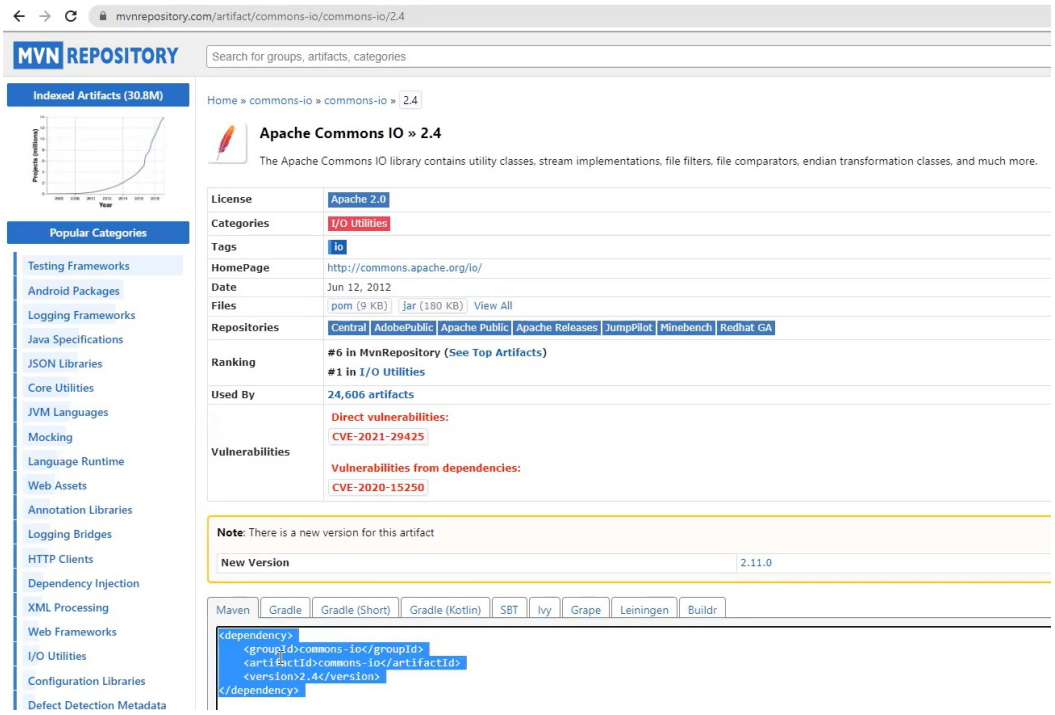
2. Create JSON content from Java Object

File Utils class

FileUtils class is an easy way to work with files. It allow you to read, write, copy the files.

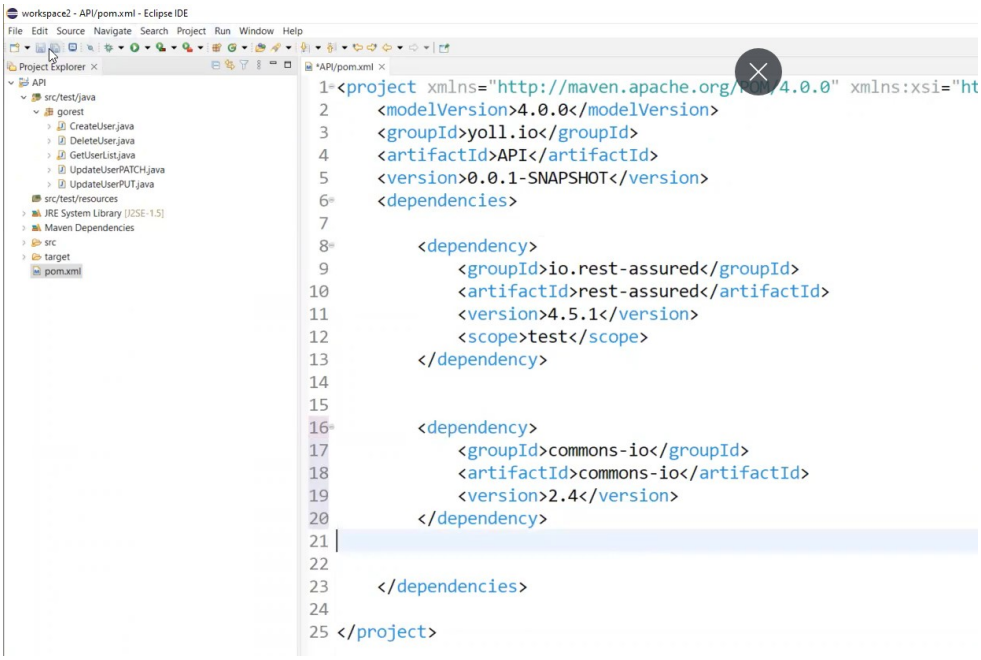
In order to use FileUtils class we need to add its dependency to pom.xml file

```
<!-- commons-io dependency is used to work with files  -->

<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

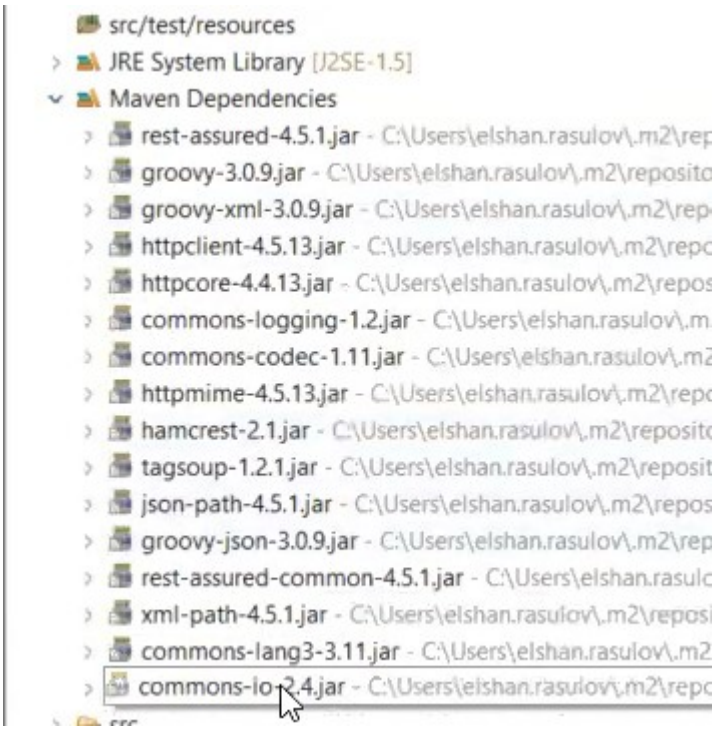


Adding Commons IO Repository to pom.xml

CHOOSE 2.4 Version !!



Save it, and make sure that the JAR File was added.



FileUtils is a utility class provided by the **Apache Commons IO** library that simplifies file and directory operations in Java. It provides methods for reading, writing, copying, moving, deleting, and comparing files, as well as working with directories. It is widely used for handling file-related tasks in Java applications.

Key Features of FileUtils

1. **File Operations**:
    - Read and write files.
    - Copy, move, and delete files.
    - Compare files.
2. **Directory Operations**:
    - Create, copy, move, and delete directories.
    - List files in a directory.
3. **File Content Handling**:
    - Read file content as a String or byte[].
    - Write String or byte[] content to a file.
4. **Error Handling**:
    - Provides methods that handle exceptions gracefully.

How to Use FileUtils

1. Add Apache Commons IO Dependency

To use FileUtils, you need to add the **Apache Commons IO** library to your project.

Maven (pom.xml):

```
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.13.0</version>
</dependency>
```

Example: Reading JSON from a File for API Testing

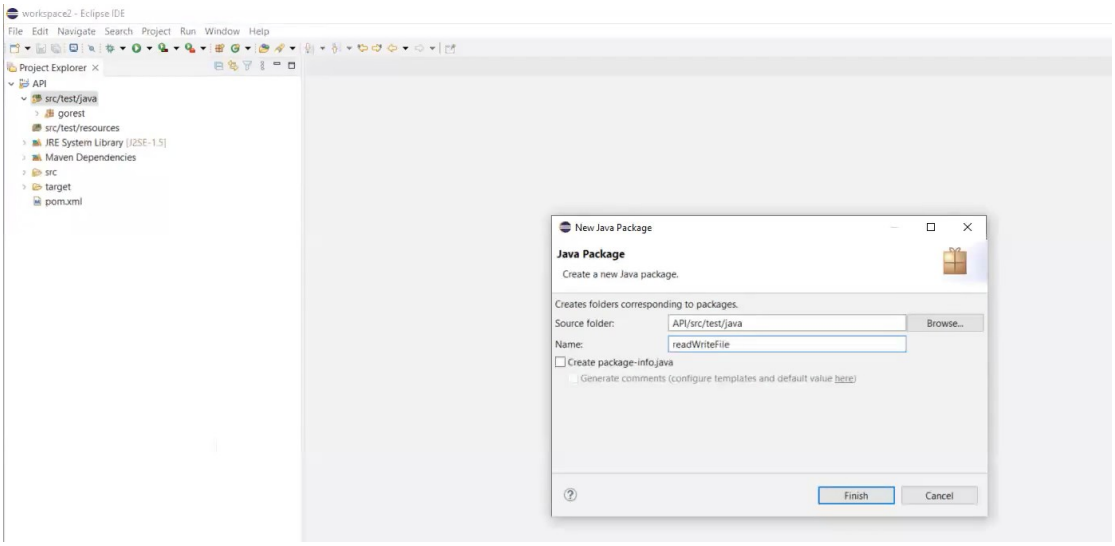
If you are using FileUtils for API testing, you can read a JSON file and use it as a request body.

```
import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.apache.commons.io.FileUtils;
import java.io.File;
import java.io.IOException;

public class ApiTestExample {
    public static void main(String[] args) {
        // Path to the JSON file
```

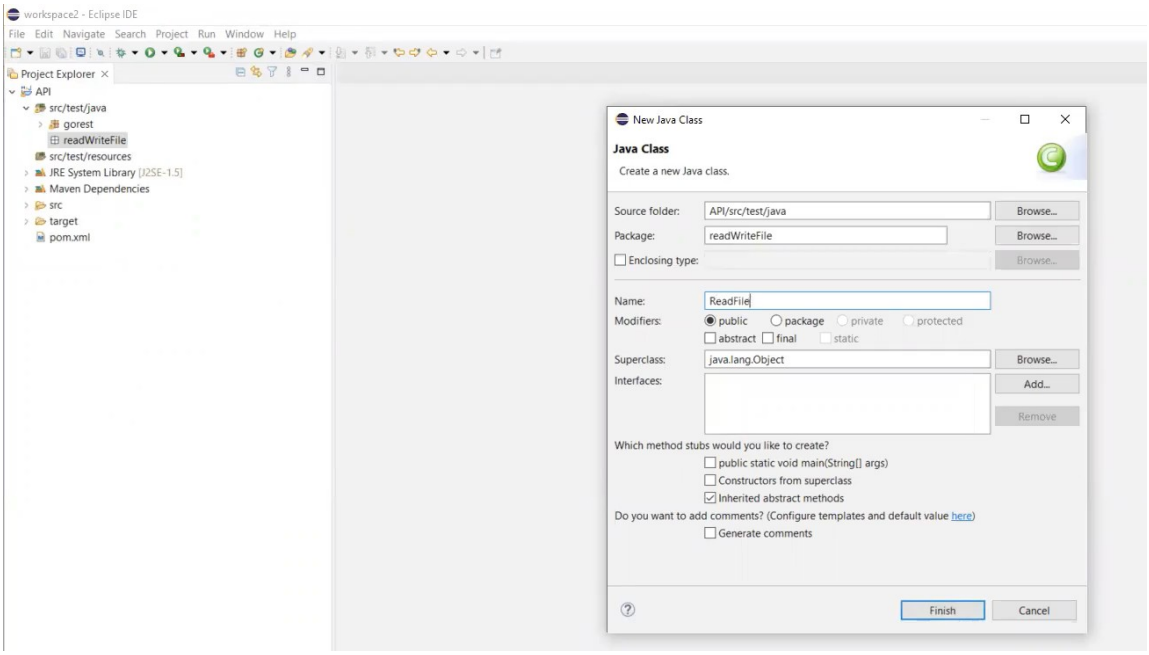
```
        File file = new File("path/to/requestBody.json");

        try {
            // Read the file content into a String
            String requestBody = FileUtils.readFileToString(file, "UTF-8");

            // Send POST request with the JSON payload
            Response response = RestAssured.given()
                    .header("Authorization", "Bearer YOUR_ACCESS_TOKEN")
                    .contentType("application/json")
                    .body(requestBody)
                    .post("https://gorest.co.in/public/v1/users");

            // Print the response
            System.out.println("Status Code: " + response.getStatusCode());
            System.out.println("Response Body: " +
response.getBody().asString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Steps to Read a file :

1- Create a package src/test/java, name it: readWriteFile



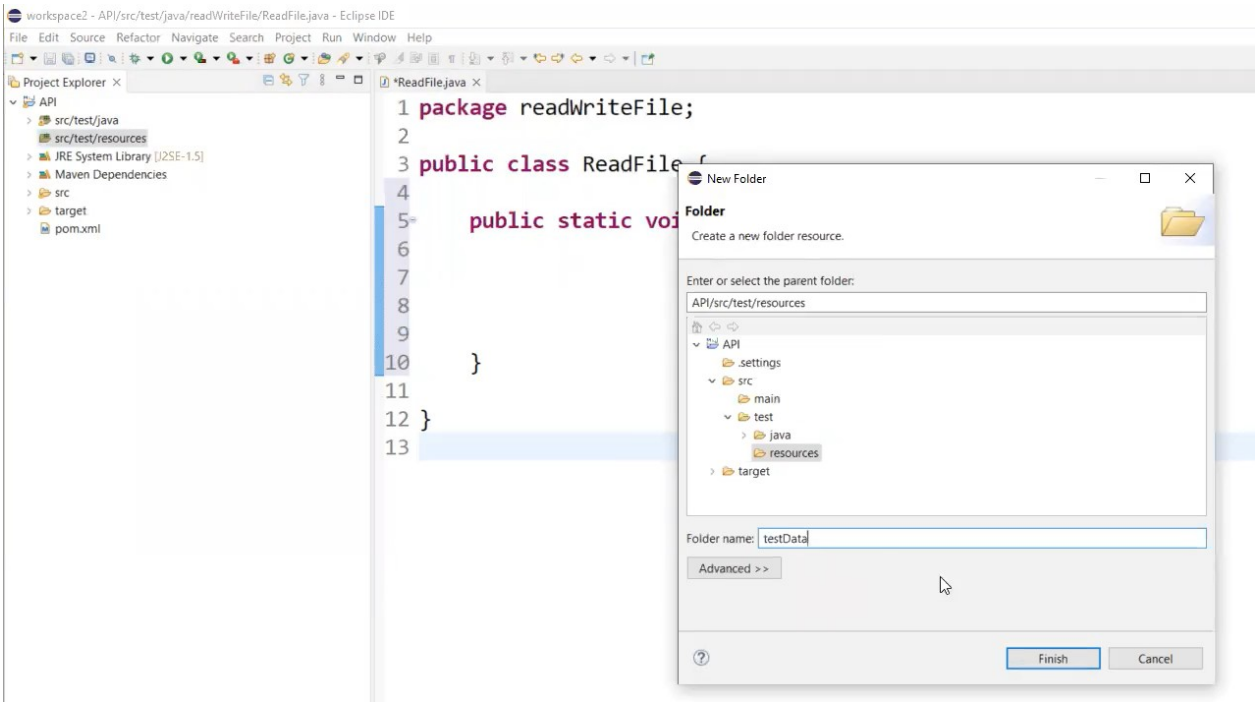
2- Create a class under the package readWriteFile, name it: ReadFile



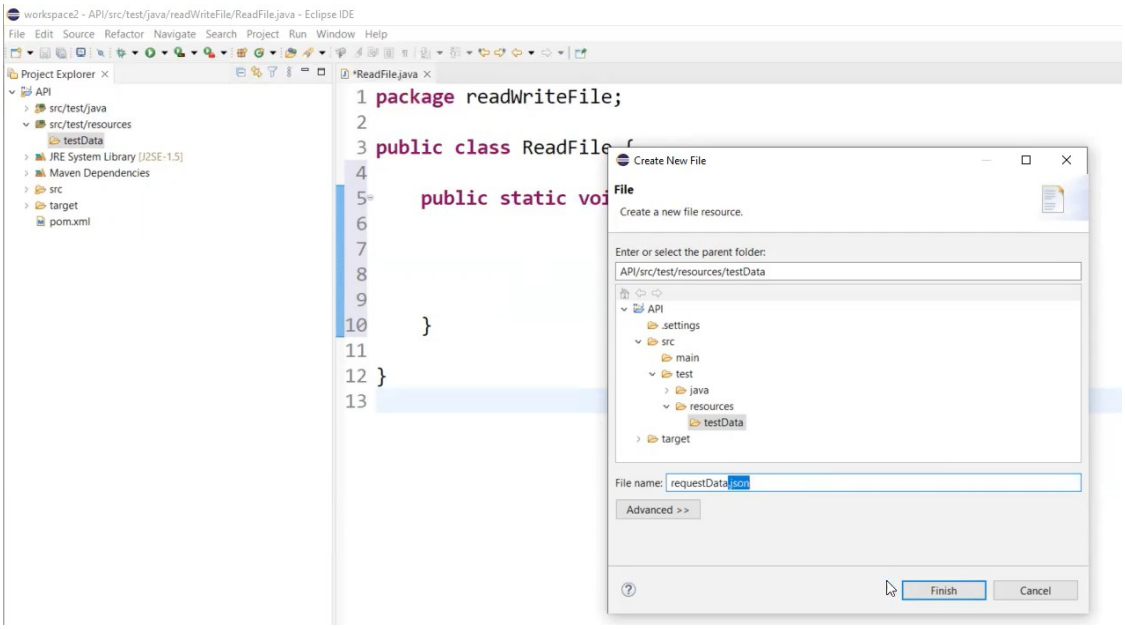
3- Create the maim method:

```
public static void main(String[] args) {

}
```

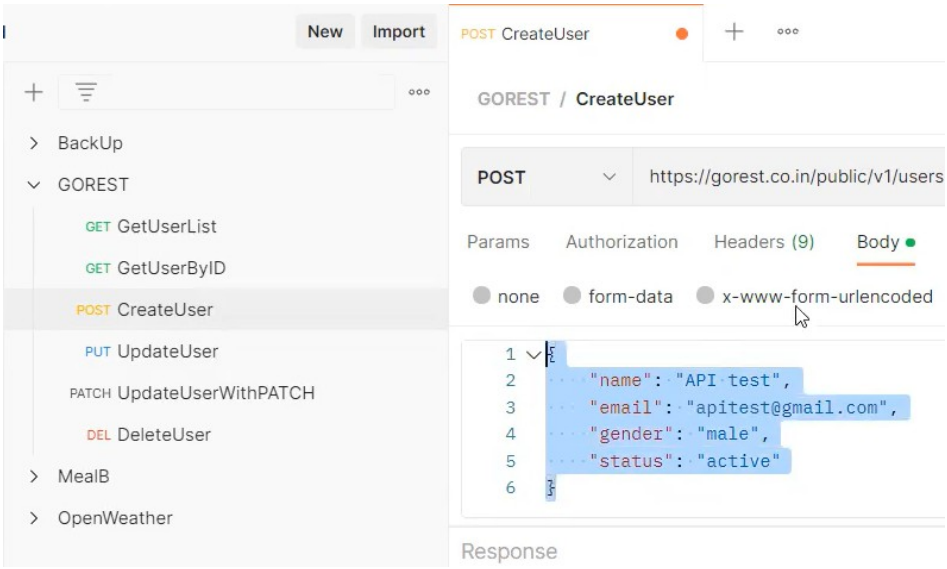
Create a folder under src/test/resources, name it : testData

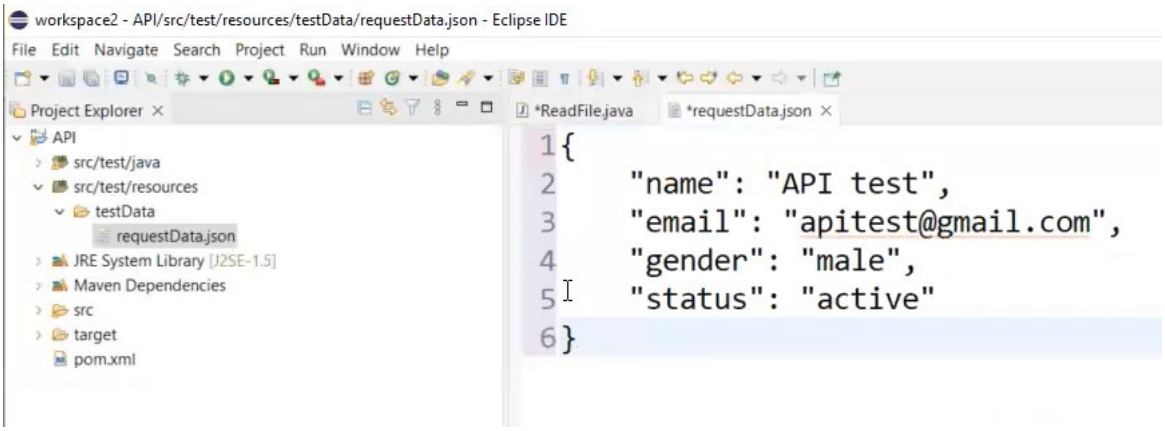


4- Create a Json file under the folder testData, name it : requestData.json



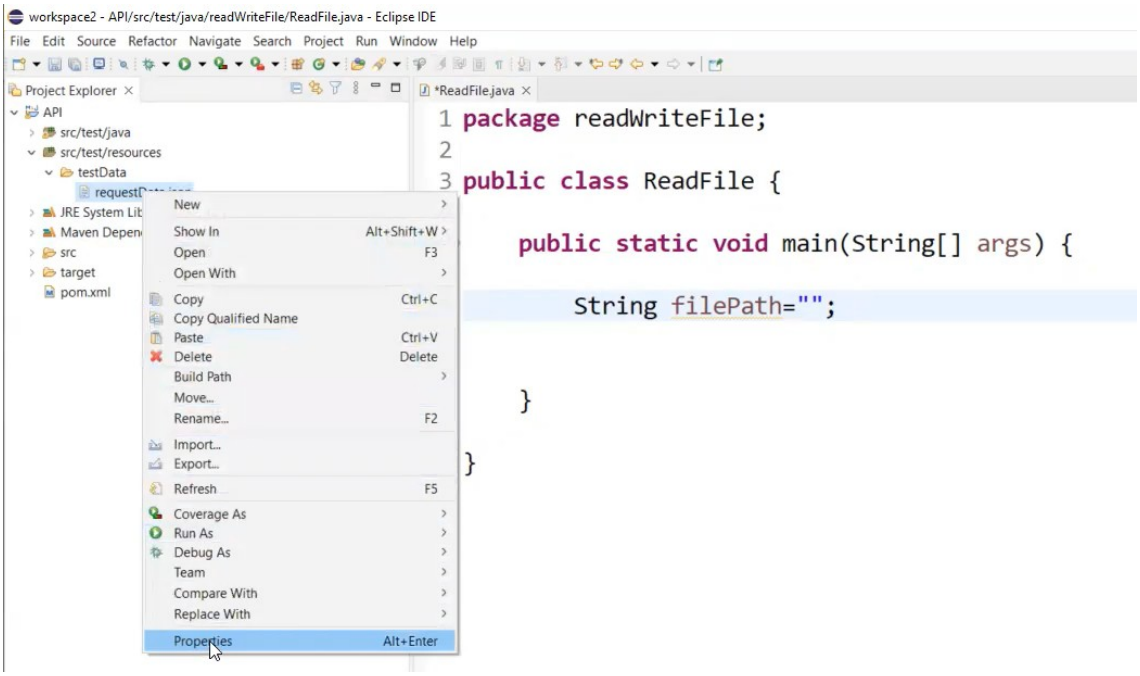
5- Copy the data on the json file:





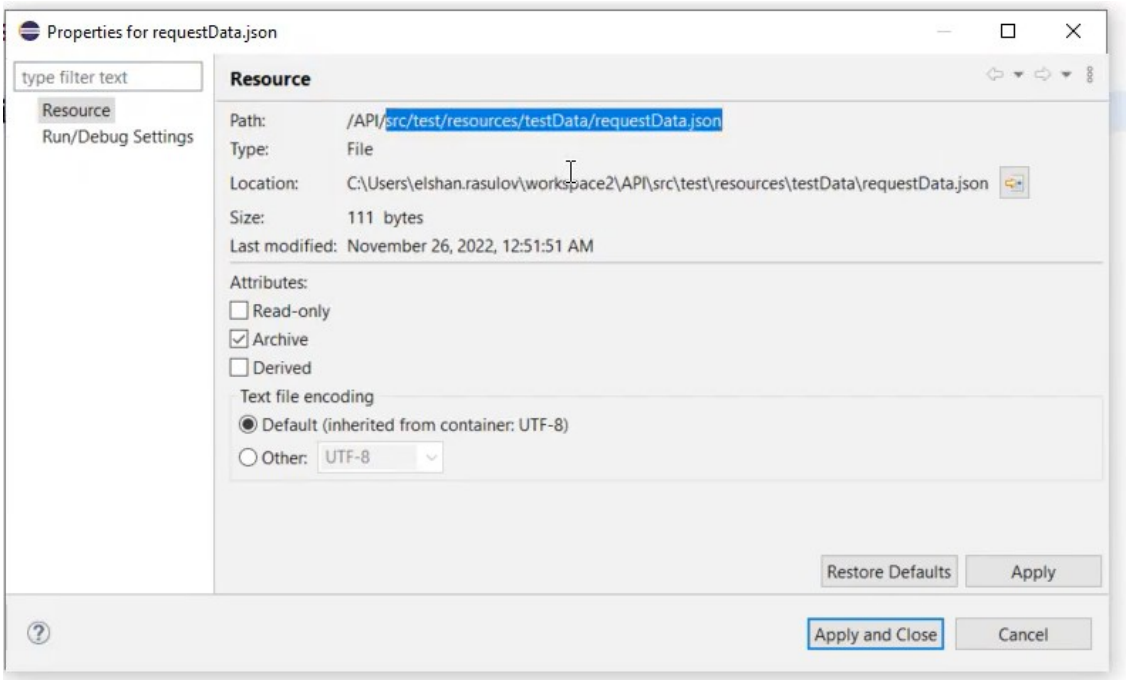
6- Save the file, and make sure that it contains the data.

7- Get the fila path, and paste it on filePath:



Cancel the Dialogue Message (Editors available on the Market Place), we do not need it.

8- Copy from Path, not Location, and starts from src/test/....




```
package readWriteFile;

public class ReadFile {

    public static void main(String[] args) {

        String filePath="src/test/resources/testData/requestData.json";

    }

}
```

9- Import File (java.io)

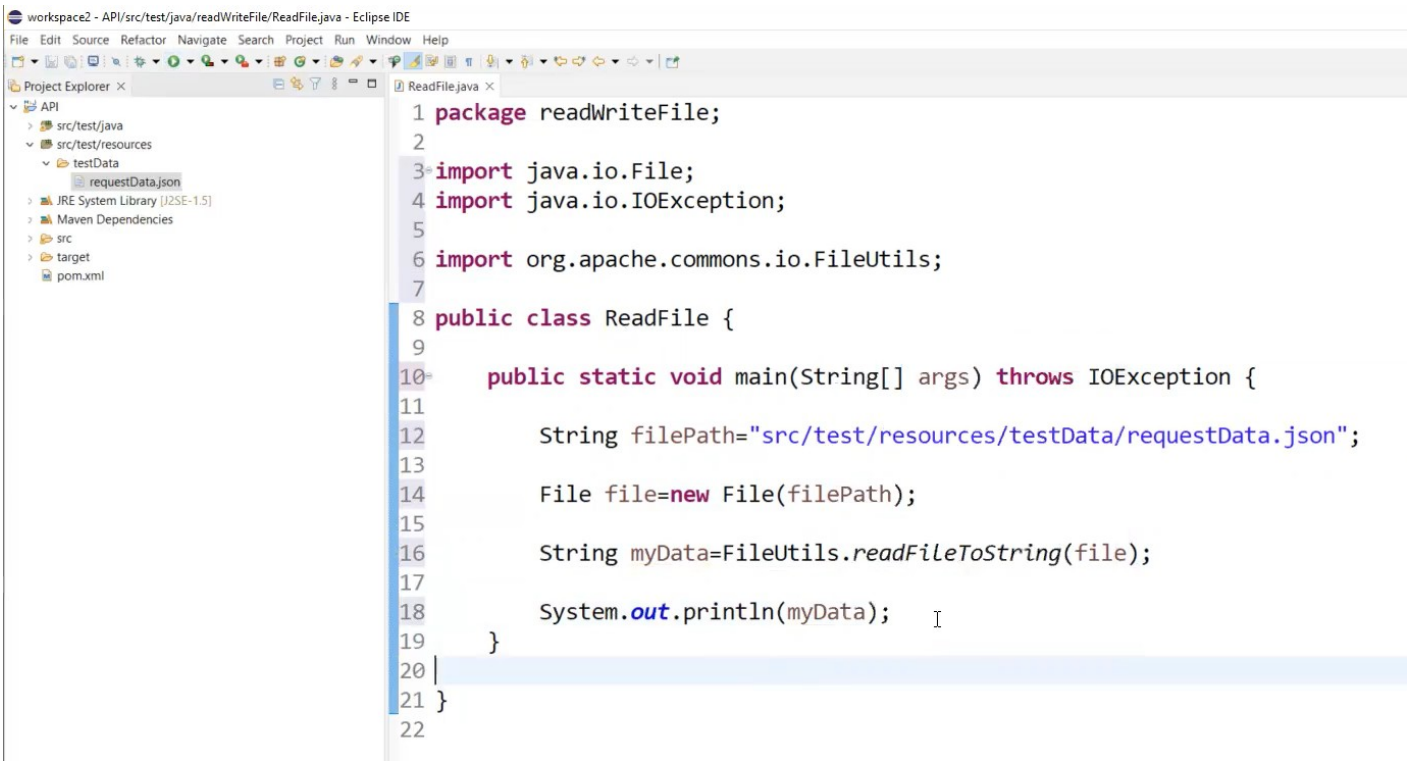
```
    public static void main(String[] args) {

        String filePath="src/test/resources/testData/requestData.json";

        File file=new File();
    }
}
```

File cannot be resolved to a type

20 quick fixes available:

- Import 'File' (java.io)
- Create class 'File'
- Create interface 'File'
- Change to 'FileBody' (org.apache.http.entity.mime.content)
- Change to 'FileLock' (java.nio.channels)
- Change to 'Files' (java.nio.file)
- Change to 'FileTime' (java.nio.file.attribute)
- Change to 'FileType' (groovy.io)

Press 'F2' for focus



If you choose a version above 2.4, use this code on java file:

String myData=FileUtils.readFileToString(file, Charset.defaultCharset());

or

String myData=FileUtils.readFileToString(file, "UTF-8");

Otherwise, you will get this stripped code:

```java
public class ReadFile {

    public static void main(String[] args) throws IOException {

        String filePath="src/test/resources/testData/requestData.json";

        File file=new File(filePath);

        String myData=FileUtils.readFileToString(file);

        System.out.println(myData);
    }

}
```

```java
package gorest;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;

public class ReadFile {
    public static void main(String[] args) throws IOException
{
        String
filePath="src/test/resources/testData/requestData.json";
        File file=new File(filePath);
        String mydata=FileUtils.readFileToString(file);
        System.out.println(mydata);
    }

}
```

Code Explanation

1. Define the File Path

```java
String filePath = "src/test/resources/testData/requestData.jason";
```

- **Purpose**: Specifies the path to the file you want to read.
- **Details**:

- filePath is a String variable that holds the relative path to the file.
- The file is located in the src/test/resources/testData/ directory and is named requestData.jason.

---

2. Create a File Object

```
File file = new File(filePath);
```

- **Purpose**: Creates a File object that represents the file at the specified path.
- **Details**:
  - The File class in Java is used to represent files and directories.
  - The file object points to the file at the path stored in filePath.

---

3. Read the File Content

```
String mydata = FileUtils.readFileToString(file);
```

- **Purpose**: Reads the content of the file into a String.
- **Details**:
  - FileUtils.readFileToString(file) is a method from the **Apache Commons IO** library.
  - It reads the entire content of the file and returns it as a String.
  - The content is stored in the mydata variable.

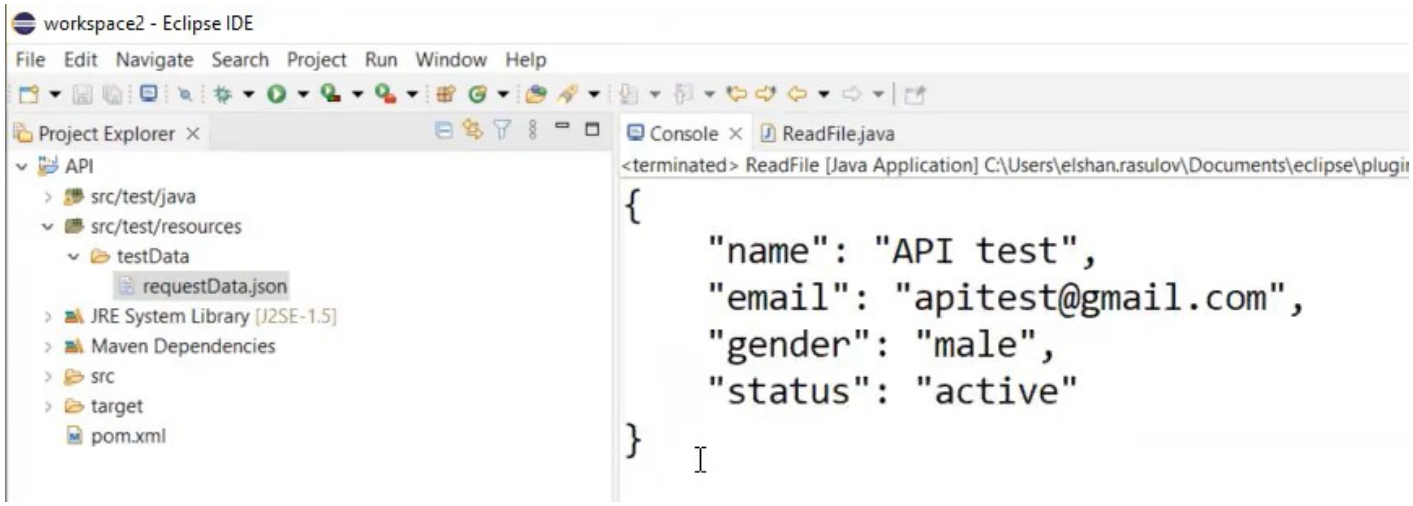
---

4. Print the File Content

```
System.out.println(mydata);
```

- **Purpose**: Prints the content of the file to the console.
- **Details**:

Result after running the code:

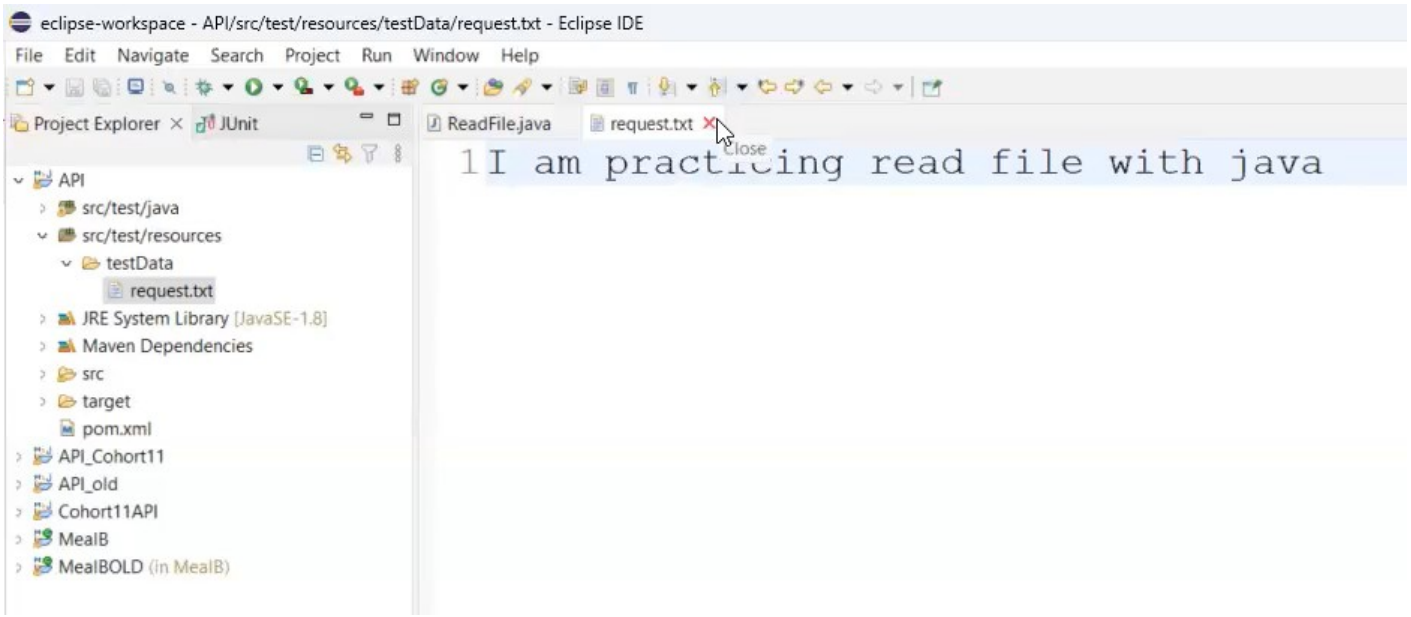

workspace2 - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
API
src/test/java
src/test/resources
testData
requestData.json
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
Console
ReadFile.java
<terminated> ReadFile [Java Application] C:\Users\elshan.rasulov\Documents\eclipse\plugi
{
"name": "API test",
"email": "apitest@gmail.com",
"gender": "male",
"status": "active"
}

Read from file using TEXT FILE: (Request)

Create a folder in ***src/test/resource*** (because it is not a java file): **TestData**.

Then create a text file (**request.txt) under** : `src/test/resources/testData`

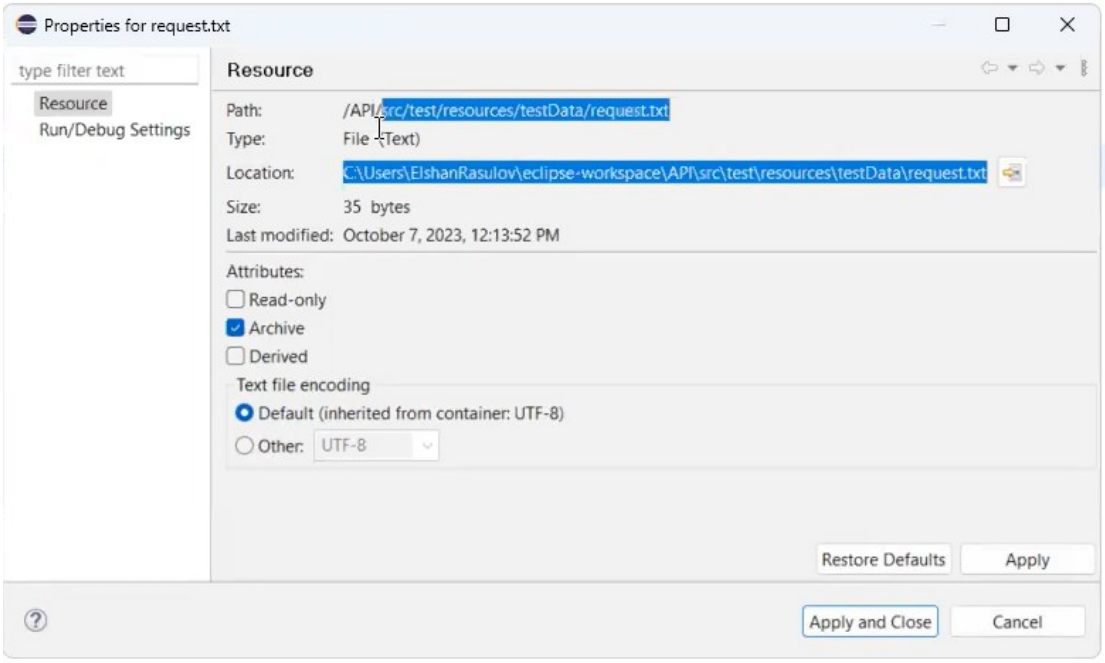
Write 'I am practicing java' **on the text file, then SAVE IT.**



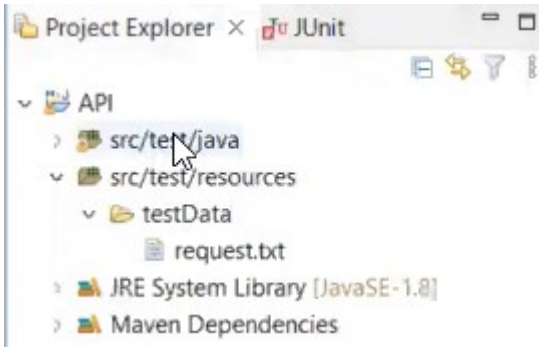
Copy the path of the text file:

Do not use this path from API: ~~**API**~~`/src/test/resources/testData/request.txt`

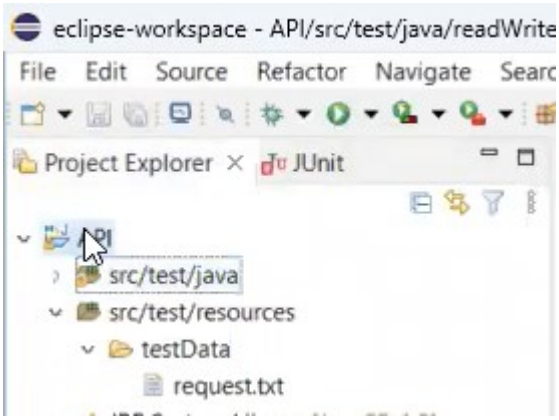
But use the path from src : `src/test/resources/testData/request.txt`



Because Java File and Text File must be on the same level



#

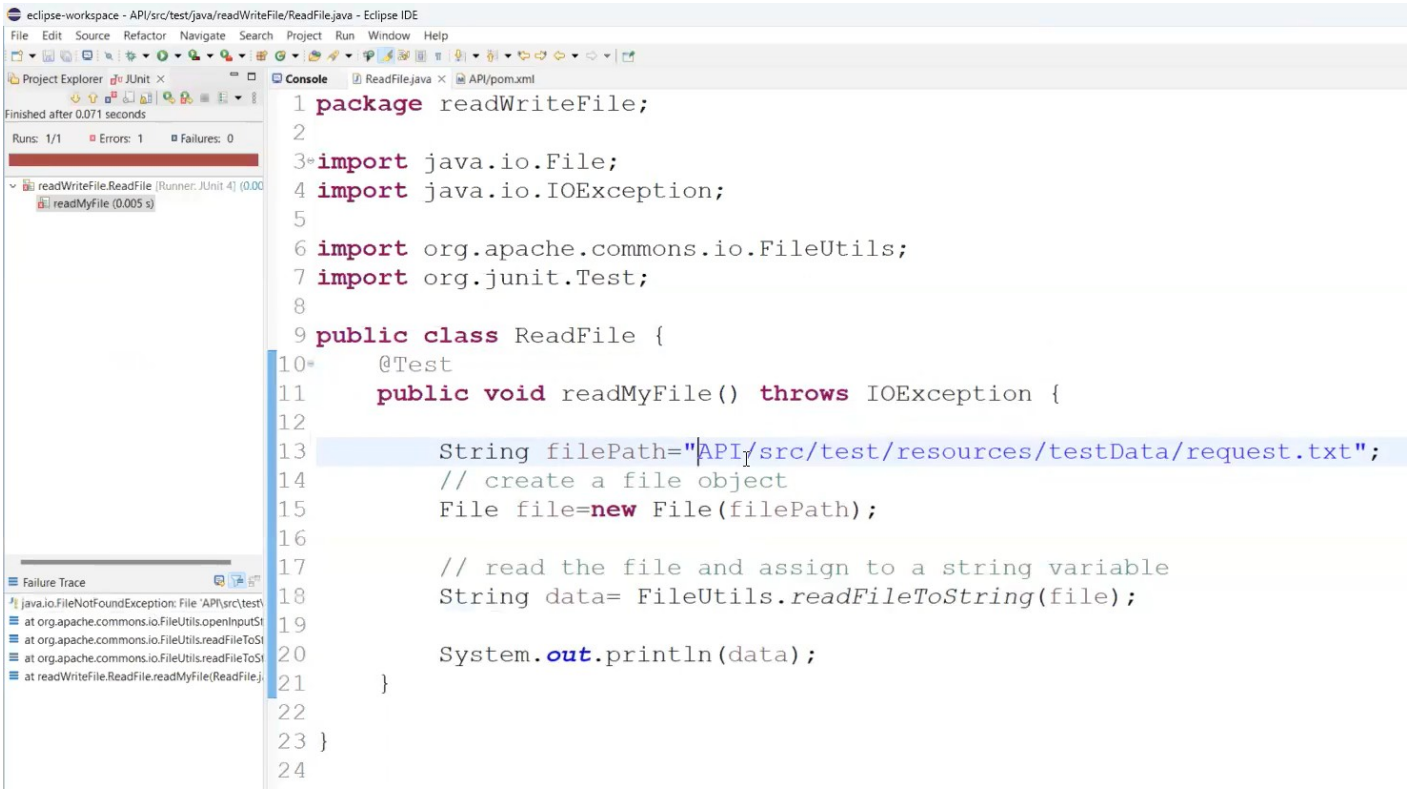


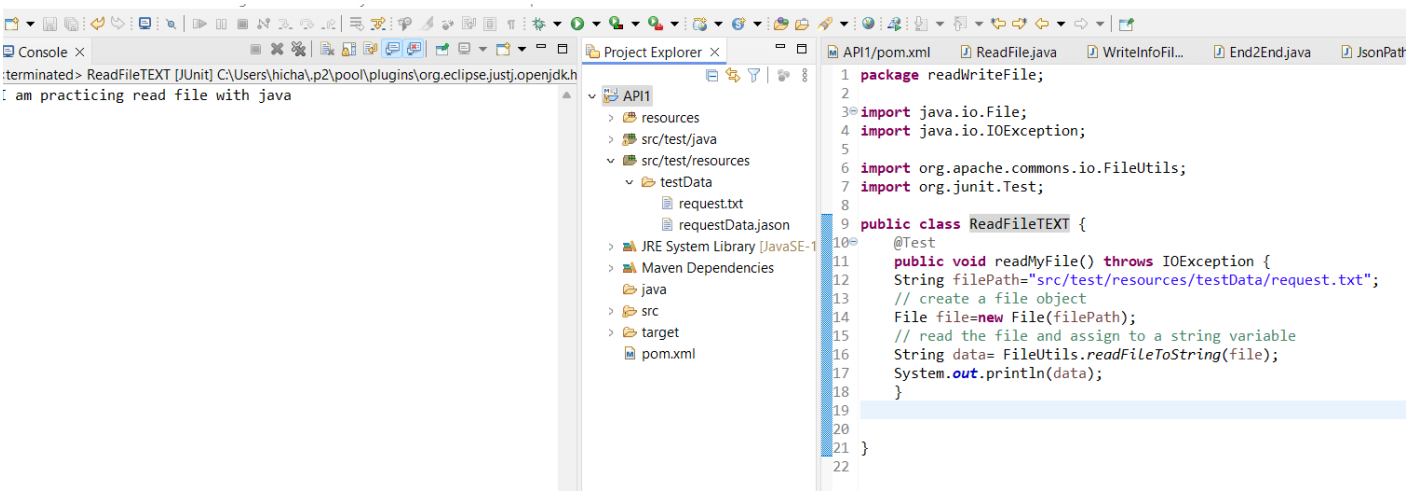
After writing the code¸ we need to add an exception because there is an external file.

```
package readWriteFile;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.junit.Test;

public class ReadFileTEXT {
     @Test
     public void readMyFile() throws IOException {
     String filePath="src/test/resources/testData/request.txt";

     // create a file object
     File file=new File(filePath);
```

```
        // read the file and assign to a string variable
        String data= FileUtils.readFileToString(file);
        System.out.println(data);
        }


}
```

What happens if we use thr path from API ?





Steps to Write a file :

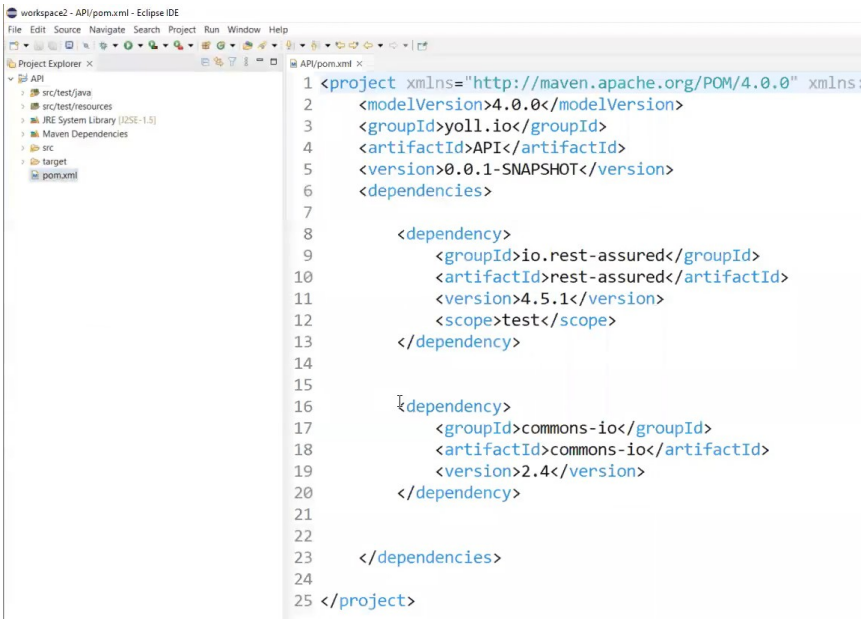
The SAME PROCESS AS READ A FILE:

- **Make sure that you have commons-io dependency on pom.xml like when we want to read a file.**
  CHOOSE 2.4 Version !! otherwise, the code will be strapped:

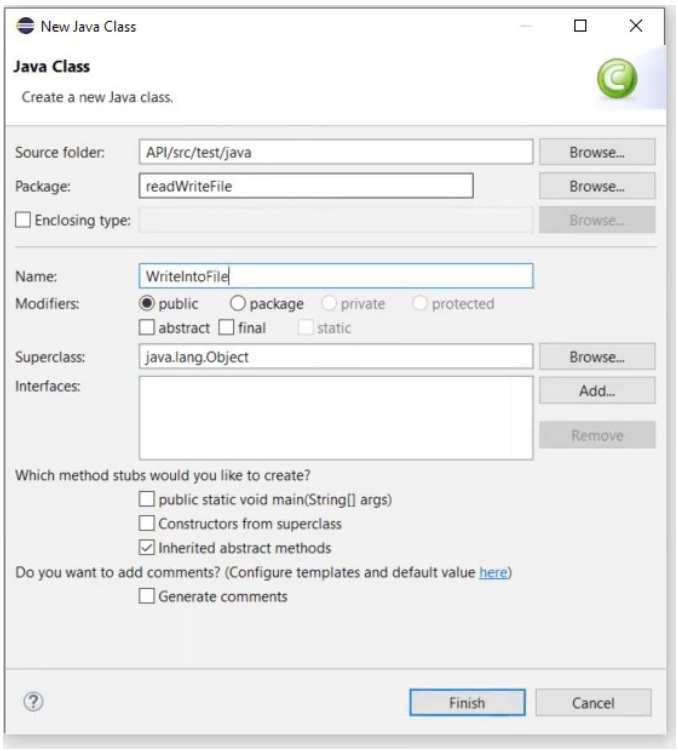
```
File file=new File(filePath);

FileUtils.writeStringToFile(file, strResponse, );
```

you need to use in this case :

```
//FileUtils.writeStringToFile(file, strResponse, Charset.defaultCharset());
FileUtils.writeStringToFile(file, strResponse, "UTF-8");
```

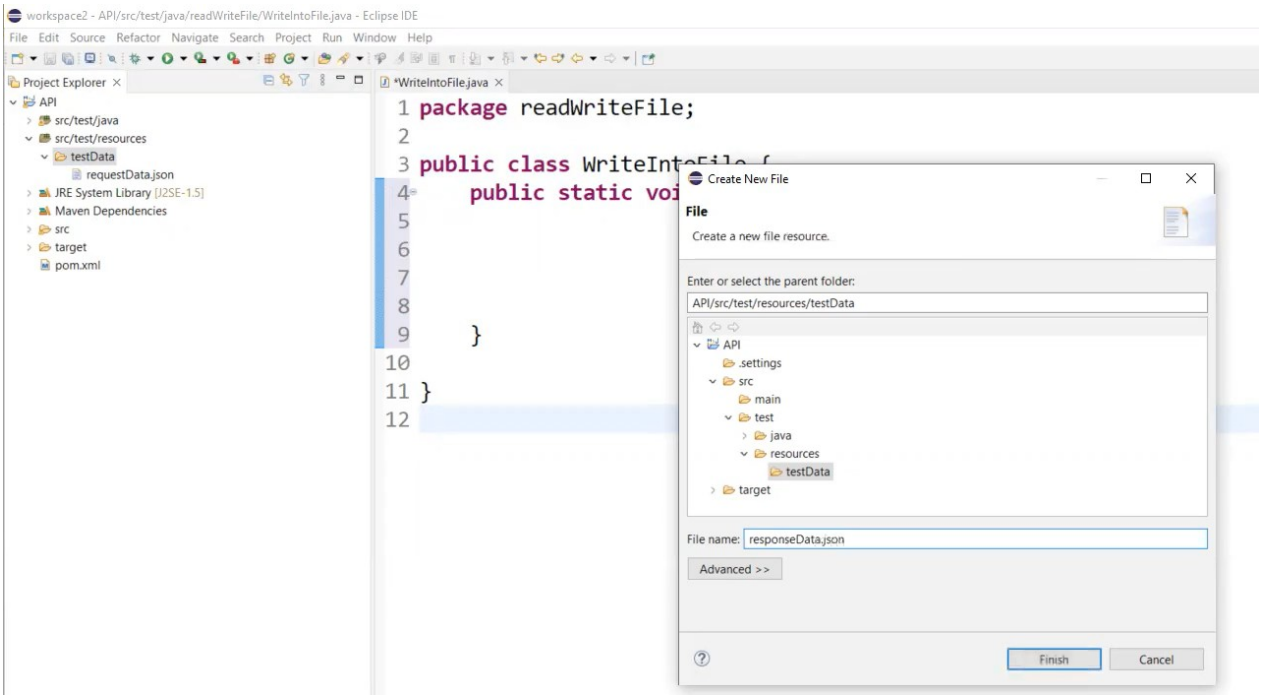


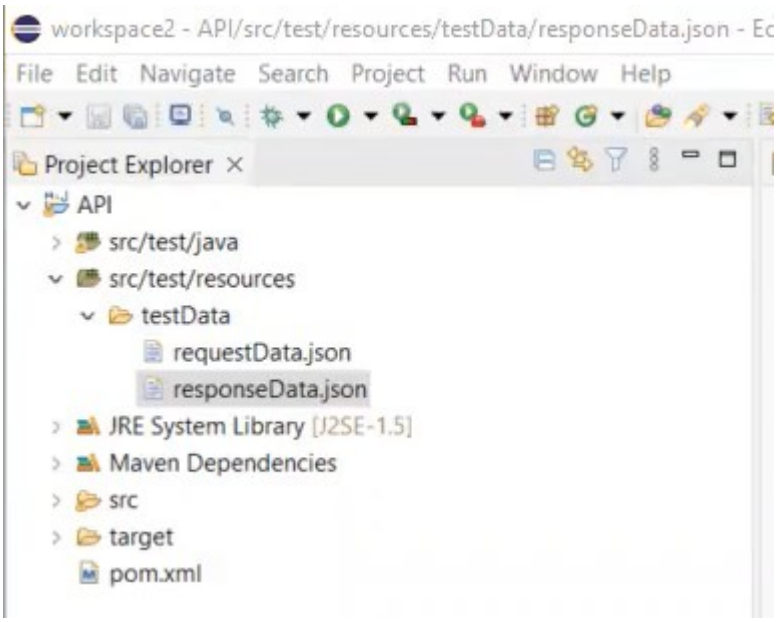
- **Create a class under the package readWriteFile, name it: WriteIntoFile**



- **Create the main method.**
- **Create a Json file (responseData.json) under the folder testData.**

Do not create it under src/test/java, but under : src/test/resources because it is not a java file !

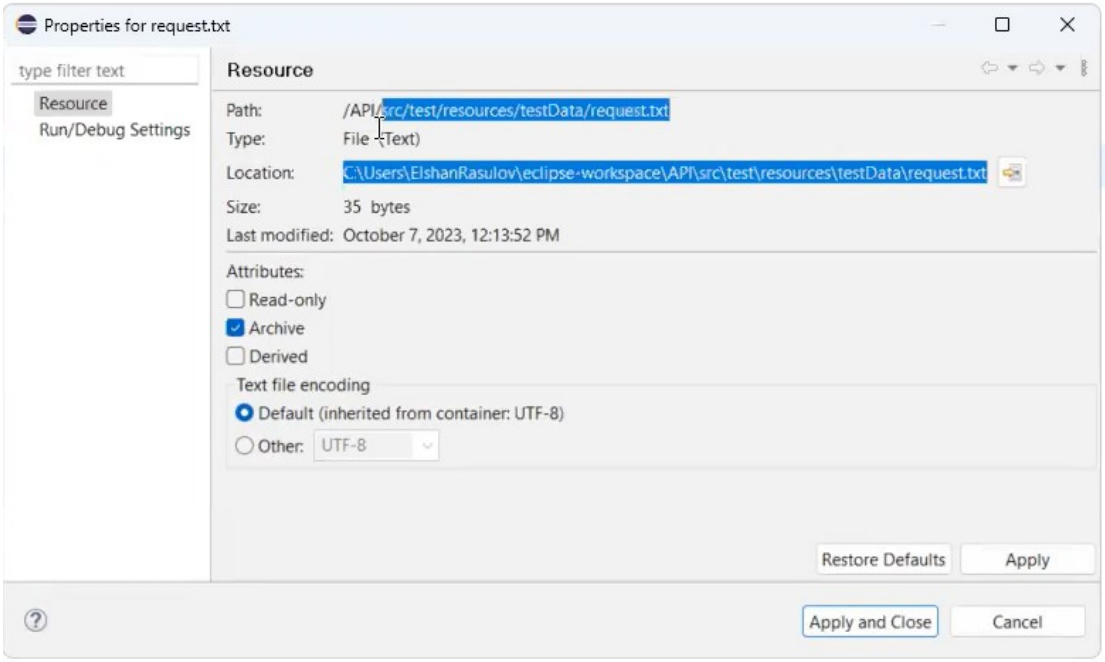


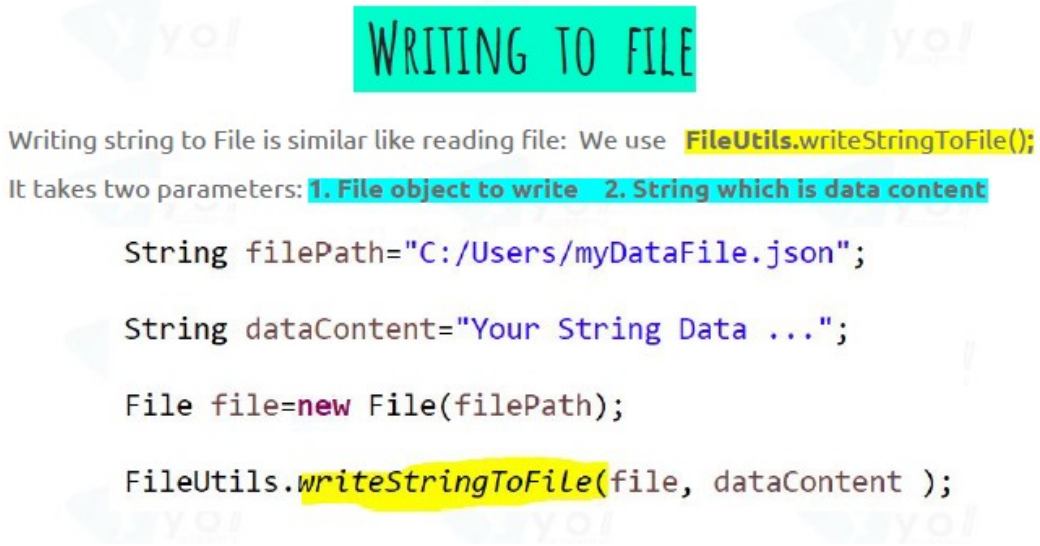


- **Copy the path of the text file:**

Do not use this path from API: ~~**API**~~`/src/test/resources/testData/request.txt`

But use the path from src : `src/test/resources/testData/request.txt`




WRITING TO FILE
Writing string to File is similar like reading file: We use FileUtils.writeStringToFile();
It takes two parameters: 1. File object to write 2. String which is data content
String filePath="C:/Users/myDataFile.json";
String dataContent="Your String Data ...";
File file=new File(filePath);
FileUtils.writeStringToFile(file, dataContent);

- **We need to write a data on the Json File**

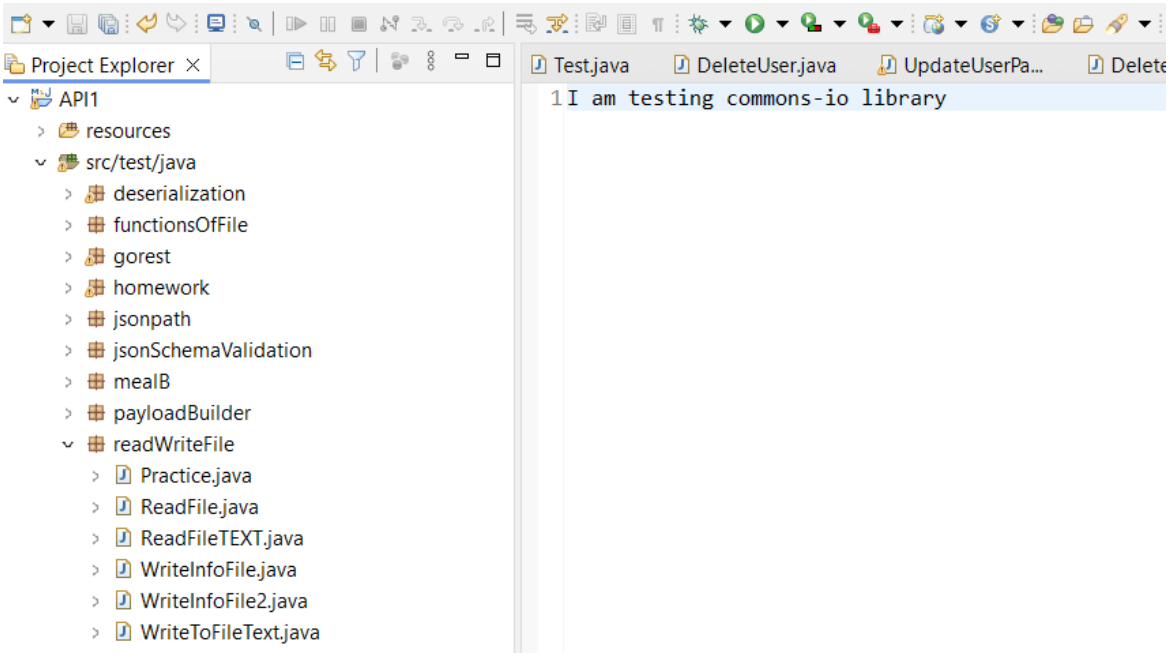
String dataContent=”Your String Data…”

- **Use FileUtils**

- **Write ="I am testing commons-io library" on `requestData.json` file.**
  assign the sentence to a String variable : myData
- **Add the exception**

```
package readWriteFile;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;

public class WriteIntoFile {
	public static void main(String[] args) throws IOException {
		String
filePath="src/test/resources/testData/requestData.json";
		String myData="I am testing commons-io library";
		File file=new File(filePath);
		FileUtils.writeStringToFile(file,myData);
	}

}
```

Result:



Task :

You have already written: ="I am testing commons-io library" on `requestData.json`.

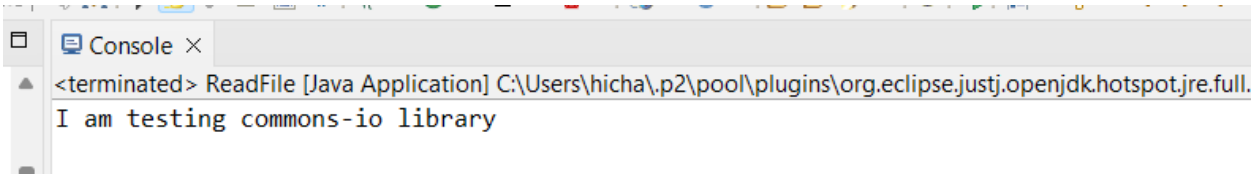
Now you need to read it:

Run the WriteIntoFile java class first,

Then run ReadFile java class

```
package readWriteFile;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;

public class ReadFile {
	public static void main(String[] args) throws IOException {
		String
filePath="src/test/resources/testData/requestData.json";
		File file=new File(filePath);
		String mydata=FileUtils.readFileToString(file);
		System.out.println(mydata);
	}

}
```

Result:



Task :

write the inactive male on the json file

```
public class WriteInfoFile2 {

	public static void main(String[] args) throws IOException {
		String url = "https://gorest.co.in/public/v1/users";

		Response response = RestAssured.given()
				.param("gender", "male")
				.param("status", "inactive")
				.when()
				.get(url);
		String strResponse=response.prettyPrint();
```

```
            String
filePath="src/test/resources/testData/requestData.json";

            File file=new File(filePath);
            FileUtils.writeStringToFile(file,strResponse);
      }

}
```

Result:


```
{
    "meta": {
        "pagination": {
            "total": 674,
            "pages": 68,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    },
    "data": [
        {
            "id": 6952773,
            "name": "Fr. Bilwa Dutta",
            "email": "fr_dutta_bilwa@champlin-schroeder.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952767,
            "name": "The Hon. Amarnath Tagore",
            "email": "hon_amarnath_tagore_the@russel-jast.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952765,
            "name": "Chandak Deshpande",
            "email": "deshpande_chandak@mann-littel.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952764,
            "name": "Bhudev Gupta",
            "email": "gupta_bhudev@mayer-walter.test",
            "gender": "male",
            "status": "inactive"
        },
        {
```

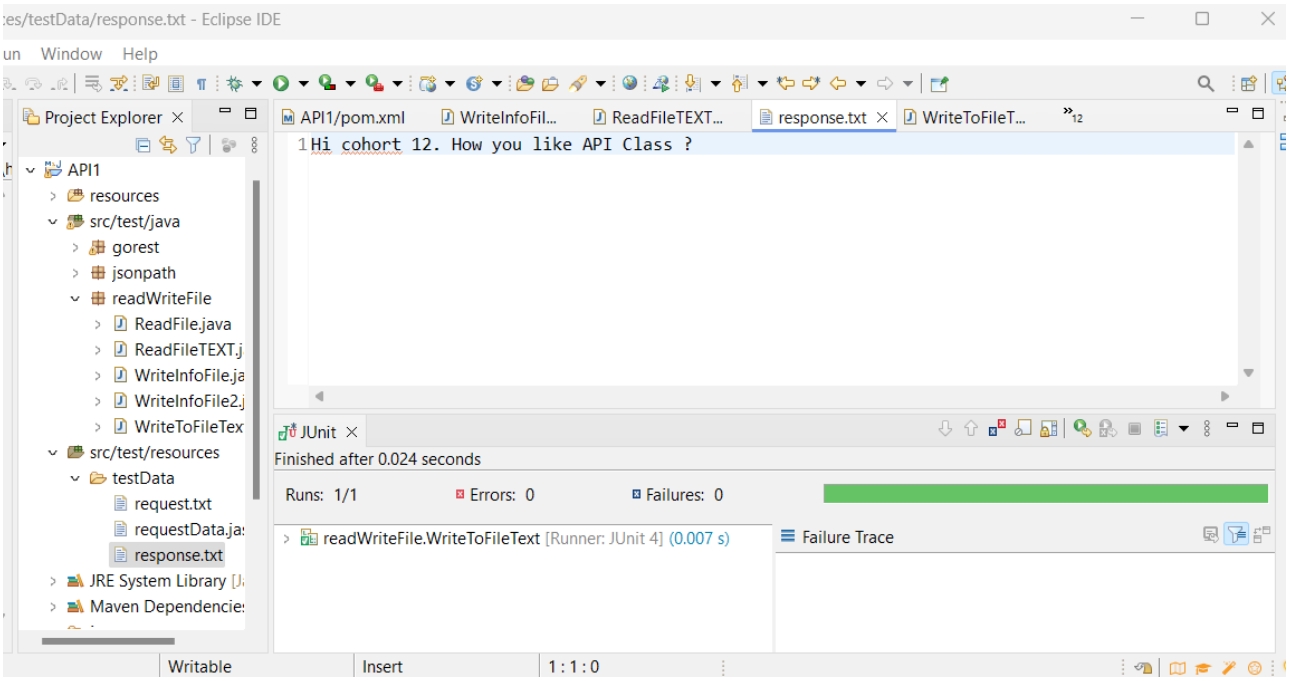
```
            "id": 6952763,
            "name": "Tara Banerjee DC",
            "email": "banerjee_dc_tara@feeney.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952761,
            "name": "Dhanapati Pilla VM",
            "email": "dhanapati_pilla_vm@mayert-witting.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952756,
            "name": "Nawal Mishra",
            "email": "nawal_mishra@runolfsson-murphy.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952755,
            "name": "Aanjaneya Khanna",
            "email": "aanjaneya_khanna@mueller-watsica.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952754,
            "name": "Chitrangada Kapoor",
            "email": "chitrangada_kapoor@lynch-steuber.example",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6952750,
            "name": "Yoginder Bhat",
            "email": "bhat_yoginder@little.example",
            "gender": "male",
            "status": "inactive"
        }
    ]
}
```


Write using TEXT FILE: (Response)

```
package readWriteFile;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.junit.Test;

public class WriteToFileText {
     @Test
```

```
    public void writeDataToFile() throws IOException {
        String data = "Hi cohort 12. How you like API Class ?";

        String filePath = "src/test/resources/testData/response.txt";
        // create file object from file path:
        File file = new File(filePath);
        // Write Data to file:
        FileUtils.writeStringToFile(file, data);

    }
}
```

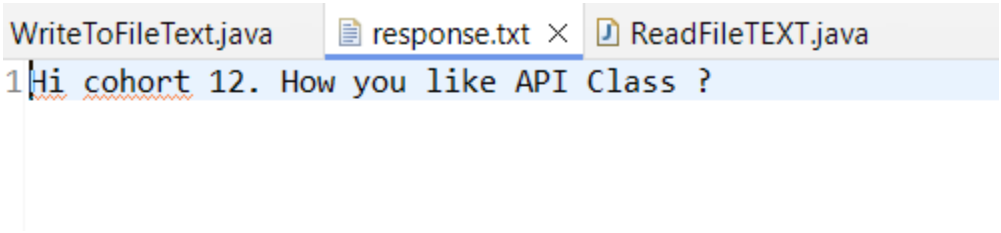
Then open the response.txt file to see the result



Read the previous file using *readFileToString*:

```
package readWriteFile;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.junit.Test;

public class ReadFileTEXT {
    @Test
    public void readMyFile() throws IOException {
```

```
        String filePath="src/test/resources/testData/response.txt";

        // create a file object
        File file=new File(filePath);
        // read the file and assign to a string variable
        String data= FileUtils.readFileToString(file);
        System.out.println(data);
        }


}
```



Practice Using gorest :

```
package readWriteFile;
import org.apache.commons.io.FileUtils;
import org.junit.Test;
import static io.restassured.RestAssured.given;
import java.io.File;
import java.io.IOException;
import io.restassured.response.Response;
public class Practice {

// write gorest response to response file
@Test
public void practice() throws IOException {

// submit request to Gorest API
Response resp=given()
.when()
.get("https://gorest.co.in/public/v1/users");

//Create a file object
String filePath="src/test/resources/testData/response.txt";
File file=new File(filePath);

//Write response to file
FileUtils.writeStringToFile(file, resp.asPrettyString());
}
}
```

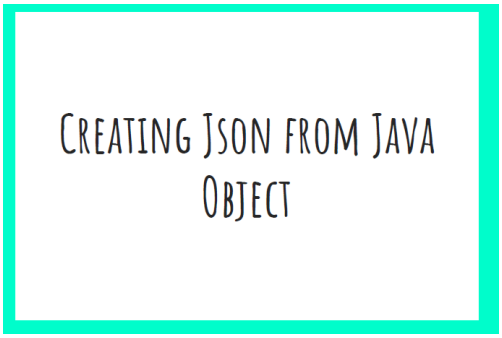
```
//asString and asPrettyString

//using asString(), will shows the RESULT ON ONE LINE, IT IS BETTER TO CHOOSE
asPrettyString().
//FileUtils.writeStringToFile(file, resp.asString());
```

RESULT :


```
{
    "meta": {
        "pagination": {
            "total": 2886,
            "pages": 289,
            "page": 1,
            "limit": 10,
            "links": {
                "previous": null,
                "current": "https://gorest.co.in/public/v1/users?page=1",
                "next": "https://gorest.co.in/public/v1/users?page=2"
            }
        }
    },
    "data": [
        {
            "id": 6961071,
            "name": "Rohan Dwivedi",
            "email": "rohan_dwivedi@douglas.example",
            "gender": "female",
            "status": "inactive"
        },
        {
            "id": 6961070,
            "name": "Swara Bandopadhyay",
            "email": "bandopadhyay_swara@medhurst.test",
            "gender": "female",
            "status": "inactive"
        },
        {
            "id": 6961069,
            "name": "Udai Sinha",
            "email": "sinha_udai@walter.example",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6961068,
            "name": "Dayamayee Mehra",
            "email": "mehra_dayamayee@doyle.test",
            "gender": "male",
            "status": "active"
        },
        {
            "id": 6961067,
            "name": "Amb. Dulari Patil",
            "email": "patil_amb_dulari@hodkiewicz.example",
```

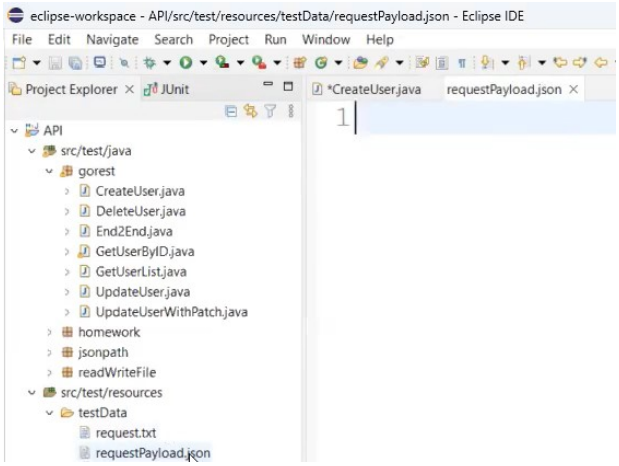
```
            "gender": "male",
            "status": "active"
        },
        {
            "id": 6961066,
            "name": "Vedanshi Embranthiri",
            "email": "embranthiri_vedanshi@wolf-hamill.example",
            "gender": "female",
            "status": "active"
        },
        {
            "id": 6961065,
            "name": "Lal Bharadwaj",
            "email": "bharadwaj_lal@braun.test",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6961064,
            "name": "Gorakhanatha Asan",
            "email": "asan_gorakhanatha@collier.example",
            "gender": "female",
            "status": "inactive"
        },
        {
            "id": 6961063,
            "name": "Rameshwar Embranthiri",
            "email": "embranthiri_rameshwar@blanda-langworth.example",
            "gender": "male",
            "status": "inactive"
        },
        {
            "id": 6961062,
            "name": "Chidaatma Shukla",
            "email": "chidaatma_shukla@armstrong.example",
            "gender": "male",
            "status": "inactive"
        }
    ]
}
```

Serialization:



is the process of converting **Java objects into JSON** (or another format like XML) so they can be sent in an **API request**. In API automation, we use serialization when sending data in **POST, PUT, or PATCH** requests. Here, we will discuss how to serialize an object to different formats such as JSON and XML using Java.

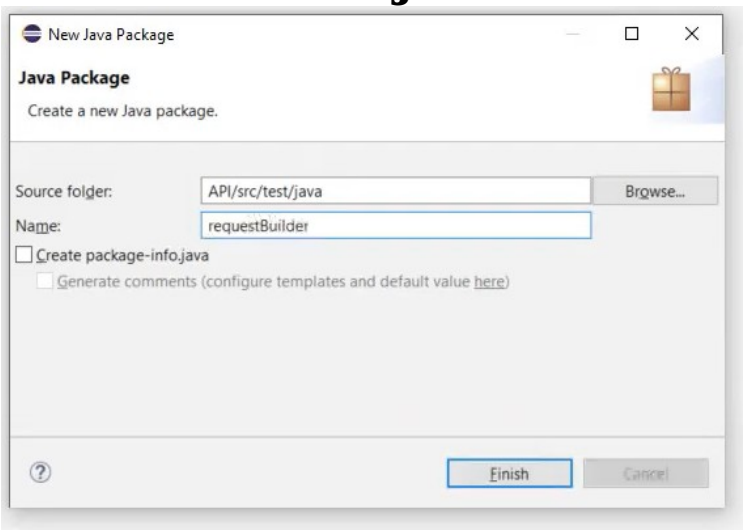
When we have a huge Data, we use an externel file: JSON File, but when it is a small data, we can use java class to put the data.



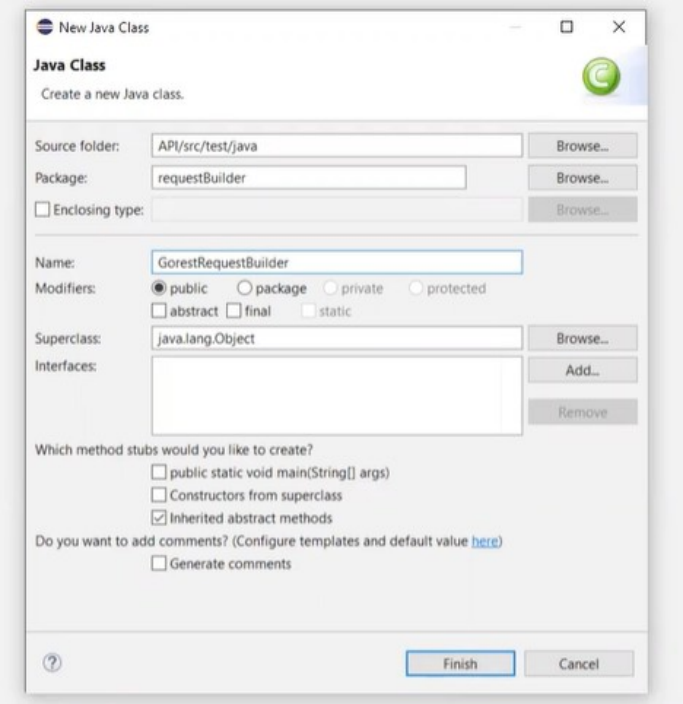

```
String payload="{\r\n"
        + "    \"name\": \"Cohort12\",\r\n"
        + "    \"email\": \"abcdefdtd1234@gmail.com\",\r\n"
        + "    \"gender\": \"male\",\r\n"
        + "    \"status\": \"active\"\r\n"
        + "}";
```

Steps:

1. **Create a package, name it: requestbuilder under src/test/java folder**



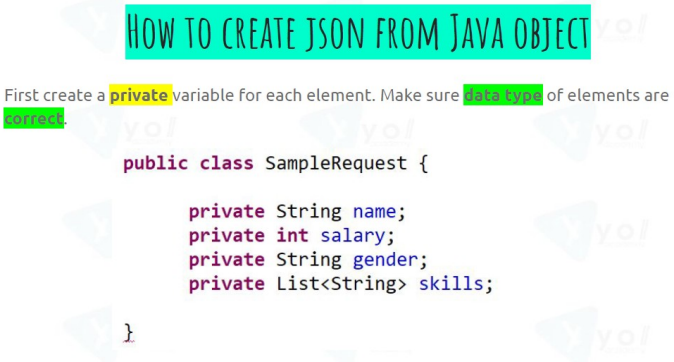
And create a class under it : **GorestRequestBuilder**



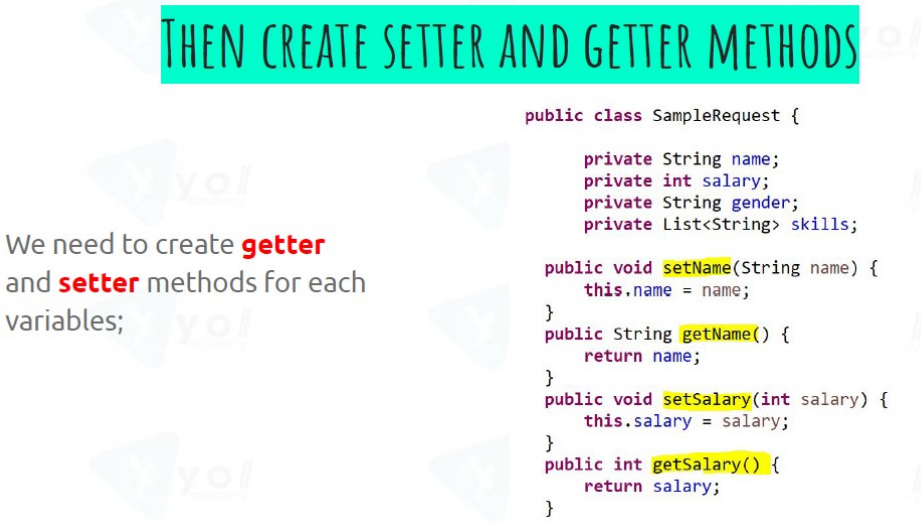
2. Create a Java class (to represent the request body.

It is a Java class with private variable, and public getter/setter methods.

- Create a private variable for each element


How to create JSON from Java object
First create a private variable for each element. Make sure data type of elements are correct.
public class SampleRequest {
private String name;
private int salary;
private String gender;
private List<String> skills;
}

- Create getters and setters



- **Create another java class: `CreateUserFromJavaObject` to call the setter methods:**

```
// Create payload data from Java object
        GorestRequestBuilder reqBuilder = new GorestRequestBuilder();
          payload.setName("Alexander Smith");
          payload.setEmail("alex34567@gmail.com");
          payload.setGender("male");
          payload.setStatus("active");
```

the Payload here is a Java Object, not a String data. It needs to be converted to String by Using Jackson Library on pom.xml, or using `ObjectMapper`.

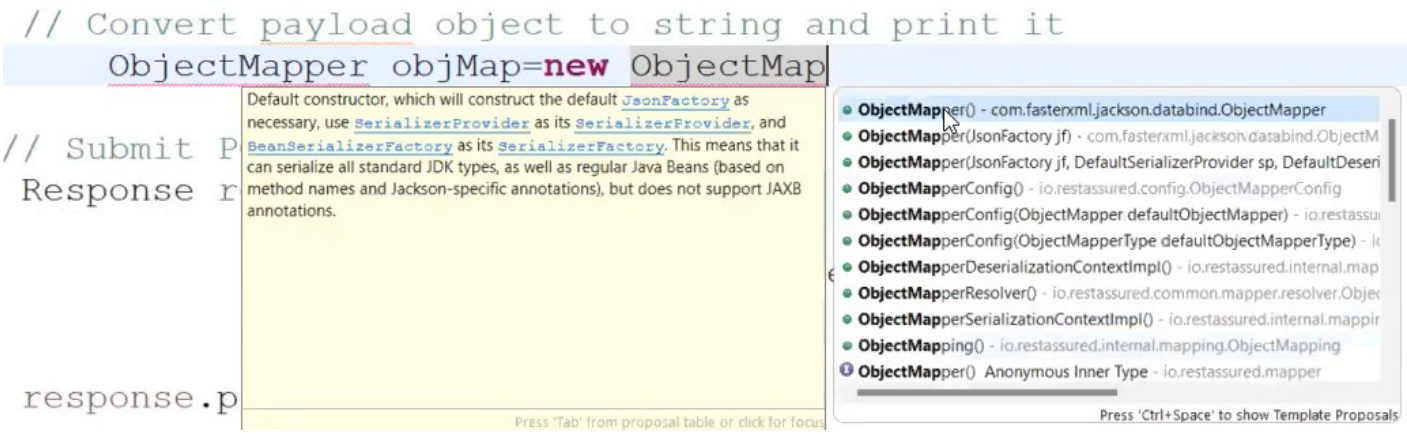
- **Use `ObjectMapper` to convert java object to String which is used for serialization and deserialization.**

`ObjectMapper` objMapper = new `ObjectMapper` ();

```
String strData = objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(reqBuilder);
        System.out.println(strData);
```

For `ObjectMapper` You must choose :

ObjectMapper()-
com.fasterxml.jackson.databind.ObjectMapper


// Convert payload object to string and print it
ObjectMapper objMap=new ObjectMap
Default constructor, which will construct the default JsonFactory as necessary, use SerializerProvider as its SerializerProvider, and BeanSerializerFactory as its SerializerFactory. This means that it can serialize all standard JDK types, as well as regular Java Beans (based on method names and Jackson-specific annotations), but does not support JAXB annotations.
ObjectMapper() - com.fasterxml.jackson.databind.ObjectMapper
ObjectMapper(JsonFactory jf) - com.fasterxml.jackson.databind.ObjectM
ObjectMapper(JsonFactory jf, DefaultSerializerProvider sp, DefaultDeseri
ObjectMapperConfig() - io.restassured.config.ObjectMapperConfig
ObjectMapperConfig(ObjectMapper defaultObjectMapper) - io.restassu
ObjectMapperConfig(ObjectMapperType defaultObjectMapperType) -
ObjectMapperDeserializationContextImpl() - io.restassured.internal.map
ObjectMapperResolver() - io.restassured.common.mapper.resolver.Objec
ObjectMapperSerializationContextImpl() - io.restassured.internal.mappir
ObjectMapping() - io.restassured.internal.mapping.ObjectMapping
ObjectMapper() Anonymous Inner Type - io.restassured.mapper
Press 'Tab' from proposal table or click for focus
Press 'Ctrl+Space' to show Template Proposals

- **This method is used with `ObjectMapper`, and add the exception:**

objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(reqBuilder).

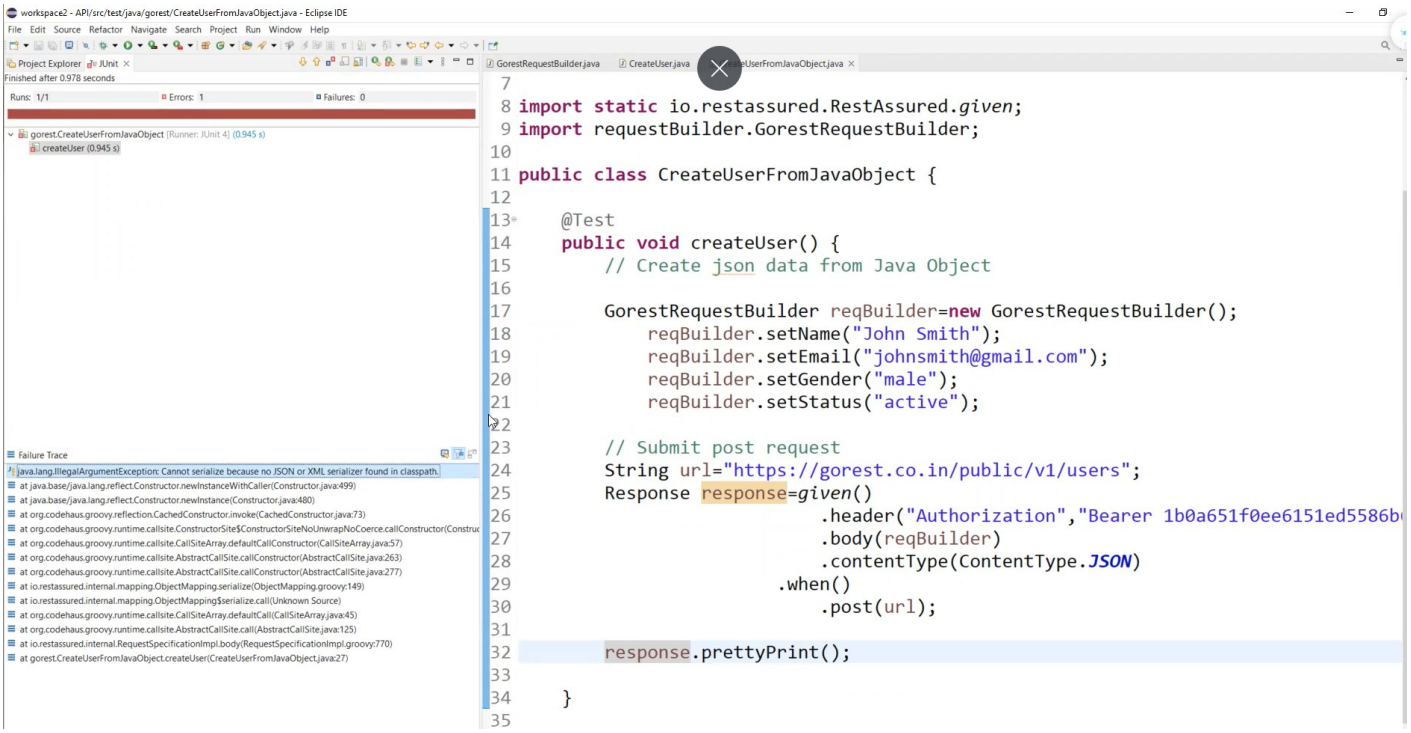
It is a part of the Jackson ObjectMapper and is used to convert a Java object to a formatted (pretty-printed) JSON string.

→ writerWithDefaultPrettyPrinter()
Enables pretty-printing, meaning the JSON output will be formatted with indentation and new lines for better readability.

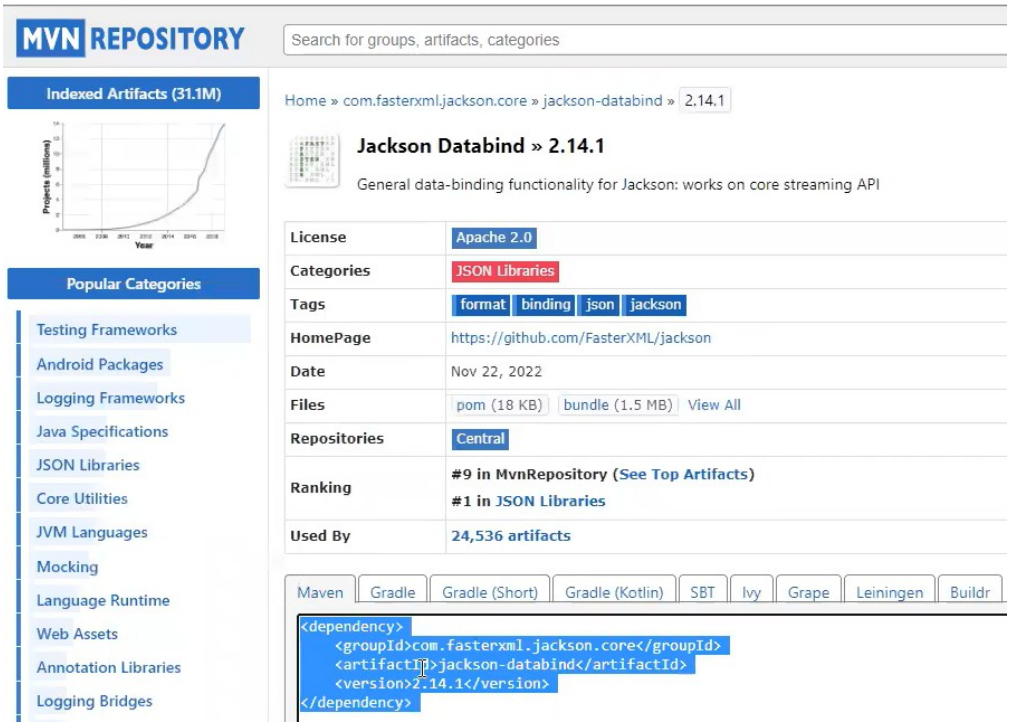
→ writeValueAsString(reqBuilder)
Serializes the reqBuilder object into a JSON string.

- **The Error is caused by not using : Jackson DataBind library.**

Cannot serialize because no Json or XML serializer found in classpath.



- **Jackson DataBind library:**




```
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
databind -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.17.1</version>
```

```
</dependency>
```

Save it, and make that the Jar file is downloaded.

Is it necessary to use `ObjectMapper` ?

It is not strictly necessary to use `ObjectMapper` to serialize an object into JSON, but it is **highly recommended** when working with APIs in Java. **And Jackson Library is enough to convert Java to Json**

✅ When is it necessary?

✅ **When you need to convert Java objects to JSON** for API requests.
✅ **When debugging JSON payloads** to see the formatted output before sending a request.
✅ **When working with REST Assured** to send complex request payloads.

❌ When is it NOT necessary?

❌ If the payload is already a String

- If you manually create the JSON payload as a string, you don't need `ObjectMapper`.
- Example:

```
String strPayload =
"{ \"name\": \"John\", \"email\": \"john@example.com\" }";
System.out.println(strPayload);
```

JSON (JavaScript Object Notation) data is a string format that represents structured data. It is commonly used for **data exchange between a client and a server in web applications.**

❓ Why is JSON a String?

1. **JSON is Text-Based** → JSON is stored and transmitted as a **text string** (UTF-8 encoding).
2. **Java Objects Must Be Serialized** → In Java, when you send JSON data in an API request, it is converted into a **string**.
3. **APIs Accept JSON as a String** → When making API calls using Java (e.g., with RestAssured), JSON data must be passed as a **string in the request body**.

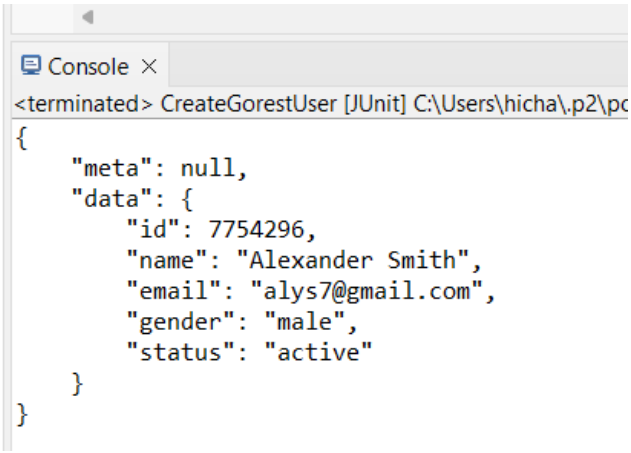
3- Sending JSON Data in an API Request:

```
//Send API request
Response response = given()
        .header("Authorization", "Bearer YOUR_ACCESS_TOKEN")
// Add authentication token
        .header("Content-Type", "application/json")
        .body(jsonRequest) // Send JSON payload
        .when()
        .post(url);
```

Example : Converting Java to Json by using only Jackson Library:

```
package serialization;

import static io.restassured.RestAssured.given;
import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GorestPayloadBuilder;

public class CreateGorestUser {
      String url = "https://gorest.co.in/public/v1/users";
      String token = "Bearer
1b0a651f0ee6151ed5586b675cd871b4178f23f1ee6900f0fb7a34d787ac1855";

      @Test
      public void testSerialization() throws JsonProcessingException {
// Create payload data from Java object
```

```
            GorestPayloadBuilder payload = new GorestPayloadBuilder();


            payload.setName("Alexander Smith");
            payload.setEmail("alys7@gmail.com");
            payload.setGender("male");
            payload.setStatus("active");
// Submit POST request to create user
            Response response = given().header("Authorization",
token).contentType(ContentType.JSON)
//.body(strPayload)
                        .body(payload).when().post(url);
            response.prettyPrint();
      }
}
```



We have two ways to use EITHER:

- STRING METHOD (strData) by putting strData in the body .body(strData),

OR

- JAVA METHOD by putting reqBuilder in the body .body(reqBuilder) TO THE BODY:

3. Use a library like Jackson to convert the Java object into JSON.

- Add Dependency

If using **Maven**, add the **Jackson** library to **pom.xml**:

```
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/
jackson-databind -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.17.1</version>
        </dependency>
```

Note: Without jackson-databind library RestAssured **body()** method **does not** accept java object.

jackson-databind library is also used to convert java object to **Json**.

4. **Send the serialized JSON in an API request.**

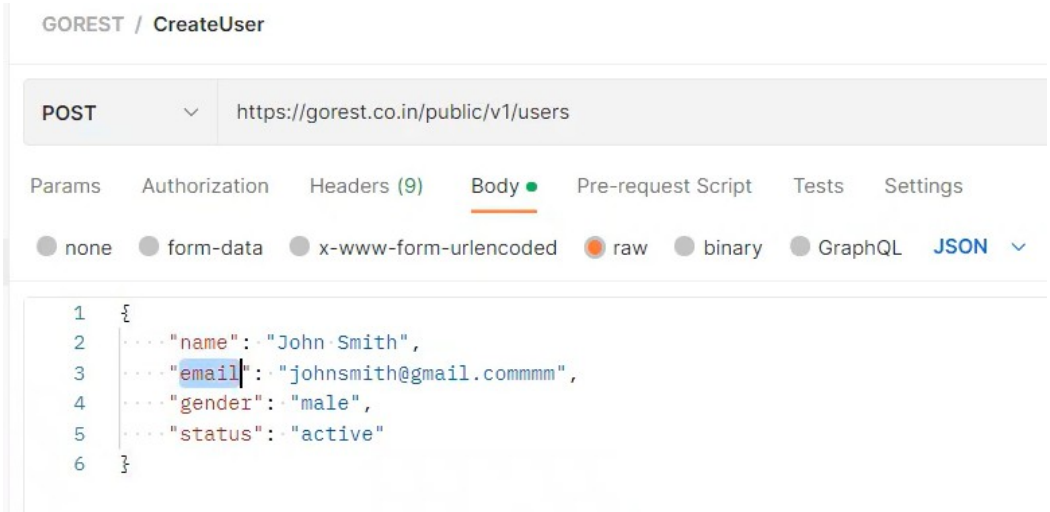
we convert the User object into JSON using Jackson's ObjectMapper

5. **Deserialize** the response (convert JSON back into Java object).

First :

Creating **GorestRequestBuilder** class with the **Getters** and **setters using the Encapsulation concept.**

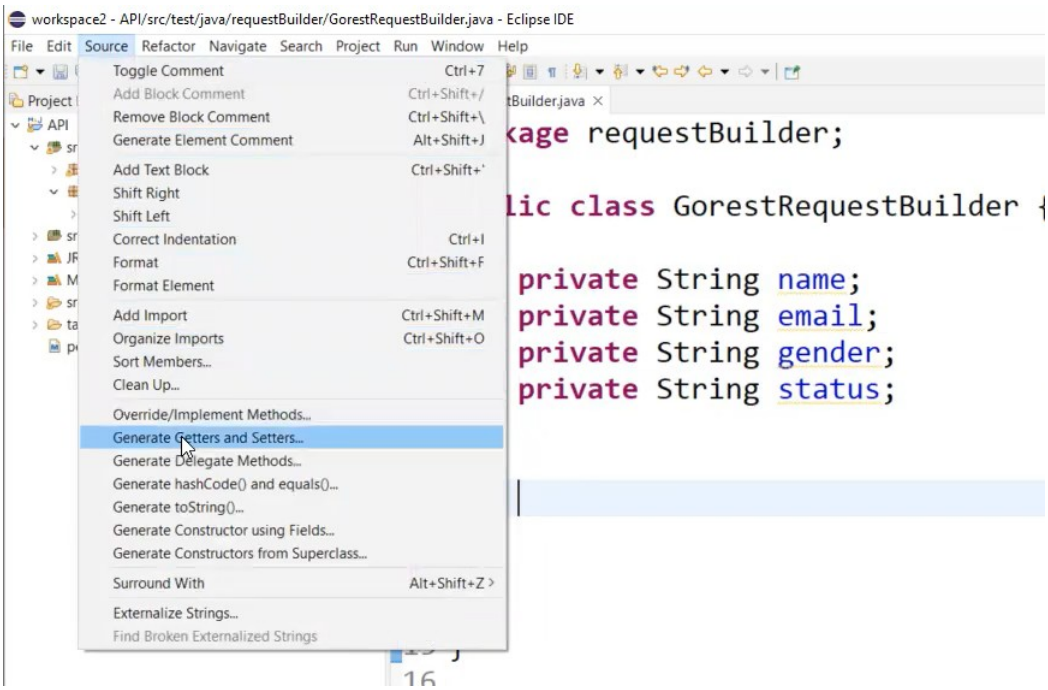
Make sure that elements are written correctly like on Postman:



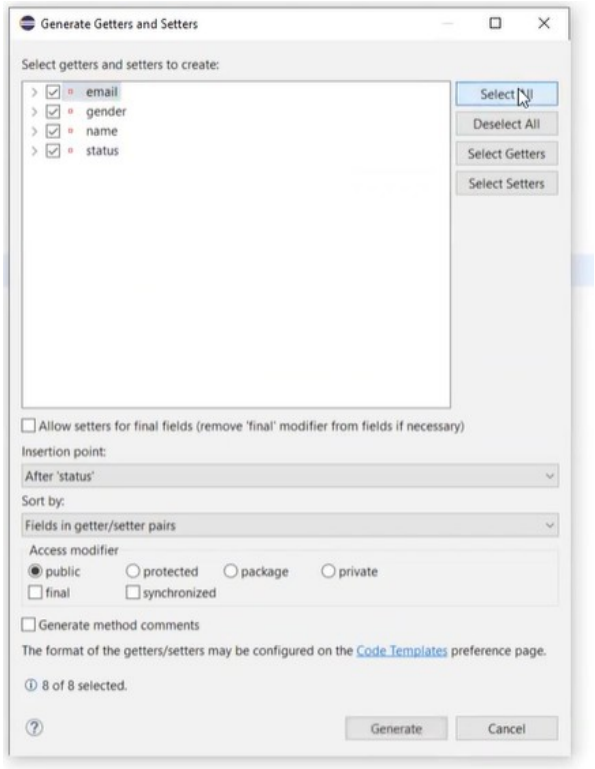
How to generate the getters and setters ?

Write the private variables first.

Go to : Source/ Generate Getters and setters



Click on Select all, click on Generate button.




```
package requestBuilder;
```

```java
public class GorestRequestBuilder {
	private String name;
	private String email;
	private String gender;
	private String status;
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getGender() {
		return gender;
	}
	public void setGender(String gender) {
		this.gender = gender;
	}
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}



}
```

Second :

Create a Class : `CreateUserFromJavaObject`, and use ObjectMapper in java Class to convert Java object to string object.

1- Create JSON data from Java Object >> JAVA FORMAT.
2- CONVERTING TO STRING FORMAT.
WE CAN PASS EITHER STRING METHOD (strData) `.body(strData)`, OR JAVA METHOD `.body(reqBuilder)` TO THE BODY:

First Way :

PASSing STRING METHOD (strData) to the body, `.body(strData)`

Example : Converting Java to Json by using ObjectMapper:

```
package gorest;
import static io.restassured.RestAssured.given;

import org.junit.Test;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
import requestBuilder.GorestRequestBuilder;

public class CreateUserFromJavaObject {
    @Test
    public void createUser() throws JsonProcessingException {

        // Create JSON data from Java Object >>

        GorestRequestBuilder reqBuilder = new GorestRequestBuilder();

        reqBuilder.setName("John Smith");
        reqBuilder.setEmail("johnsmith211@gmail.com");
        reqBuilder.setGender("male");
        reqBuilder.setStatus("active");

         //CONVERTING TO STRING FORMAT:

        ObjectMapper objMapper = new ObjectMapper();

String strData =
objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(reqBuilder
);
        System.out.println(strData);


        // Submit post request
        String url = "https://gorest.co.in/public/v1/users";

        Response response = given()
                .header("Authorization", "Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
                .contentType(ContentType.JSON)
                .body(strData)
```

```
                .when()
                .post(url);

        response.prettyPrint();

    }
}
```

RESULT 1: Using ObjectMapper

PASSING STRING METHOD (strData): BY PUTTING IT IN: .body(strData)

Second Way:Withput using ObjectMapper

JAVA FORMAT: PASSING reqBuilder in the Body: .body(reqBuilder)

```
package gorest;
import static io.restassured.RestAssured.given;

import org.junit.Test;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
import requestBuilder.GorestRequestBuilder;

public class CreateUserFromJavaObject {
    @Test
    public void createUser() throws JsonProcessingException {

        // Create JSON data from Java Object >>
```

```
        GorestRequestBuilder reqBuilder = new GorestRequestBuilder();

        reqBuilder.setName("John Smith");
        reqBuilder.setEmail("johnsmith211@gmail.com");
        reqBuilder.setGender("male");
        reqBuilder.setStatus("active");


// Submit post request
        String url = "https://gorest.co.in/public/v1/users";

        Response response = given()
                .header("Authorization", "Bearer
3b95cd9a2025eaaf1f016d8c473a80824dba5db348fbab7f38e14f6bc2b8d6d7")
                .contentType(ContentType.JSON)
                .body(reqBuilder)
                .when()
                .post(url);

        response.prettyPrint();
```

RESULT 2:

Both results are the same.

MealB Homework : Useful way for Getting Token

There are two methods:

- First Method : Without using the encapsulation method.

  One class is used. (It is not useful)

- Second Method by using the encapsulation to get the Access token:

  Two Classes are Used (Useful).

You are supposed to have Jackson Library on pom.xml to convert Java Object to String, or by using ObjectMapper, Both of them print the same result:

```
ObjectMapper objMap = new ObjectMapper();
          String strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(payload);
          System.out.println(strPayload);
```

For ObjectMapper You must choose :

ObjectMapper()-
com.fasterxml.jackson.databind.ObjectMapper

// Convert payload object to string and print it
ObjectMapper objMap=new ObjectMap
Default constructor, which will construct the default JsonFactory as necessary, use SerializerProvider as its SerializerProvider, and BeanSerializerFactory as its SerializerFactory. This means that it can serialize all standard JDK types, as well as regular Java Beans (based on method names and Jackson-specific annotations), but does not support JAXB annotations.
// Submit P
Response r
response.p
ObjectMapper() - com.fasterxml.jackson.databind.ObjectMapper
ObjectMapper(JsonFactory jf) - com.fasterxml.jackson.databind.ObjectM
ObjectMapper(JsonFactory jf, DefaultSerializerProvider sp, DefaultDeseri
ObjectMapperConfig() - io.restassured.config.ObjectMapperConfig
ObjectMapperConfig(ObjectMapper defaultObjectMapper) - io.restassu
ObjectMapperConfig(ObjectMapperType defaultObjectMapperType) - io
ObjectMapperDeserializationContextImpl() - io.restassured.internal.map
ObjectMapperResolver() - io.restassured.common.mapper.resolver.Objec
ObjectMapperSerializationContextImpl() - io.restassured.internal.mappir
ObjectMapping() - io.restassured.internal.mapping.ObjectMapping
ObjectMapper() Anonymous Inner Type - io.restassured.mapper
Press 'Tab' from proposal table or click for focus
Press 'Ctrl+Space' to show Template Proposals

- **Difference between the two Methods is the code on Payload:**

// 1ST without Object

```
String payload ="{\r\n"
          + "\"usernameOrEmailAddress\" :\"ElshanRasul\",\r\n"
          + "\"password\" : \"Elshan123\"\r\n"
          + "}";
```

//2ND create payload object

```
TokenPayLoadbuilder payload = new TokenPayLoadbuilder();
payload.setUsernameOrEmailAddress("ElshanRasul");
payload.setPassword("Elshan123");
```

First Method : Without using the encapsulation method

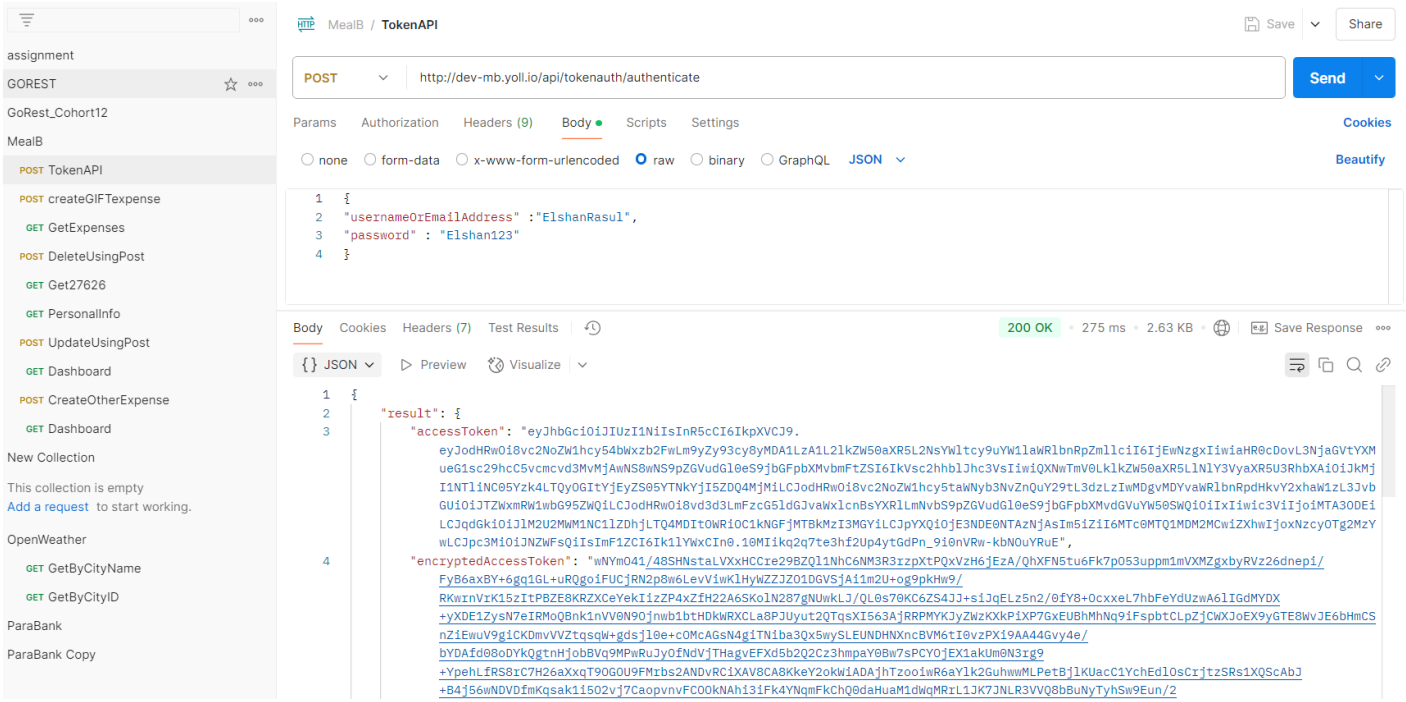
Getting the access token by using payload Java Code:

Create a Java Class : **TokenAPI**,
Then copy/paste from Body of Postman (Post request of TokenAPI) on the payload:

Code that must be on payload:

```
{
"usernameOrEmailAddress" :"ElshanRasul",
"password" : "Elshan123"
```

}




```
package mealB;

import org.junit.Test;
import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
public class TokenAPI {
String url = "http://dev-mb.yoll.io/api/tokenauth/authenticate";

String payload ="{\r\n"
          + "\"usernameOrEmailAddress\" :\"ElshanRasul\",\r\n"
          + "\"password\" : \"Elshan123\"\r\n"
          + "}";
@Test
public void getToken() {
     Response response = given()
                         .body(payload)
                         .contentType(ContentType.JSON)
                    .when()
                    .post(url);
     response.prettyPrint();
}
}
```

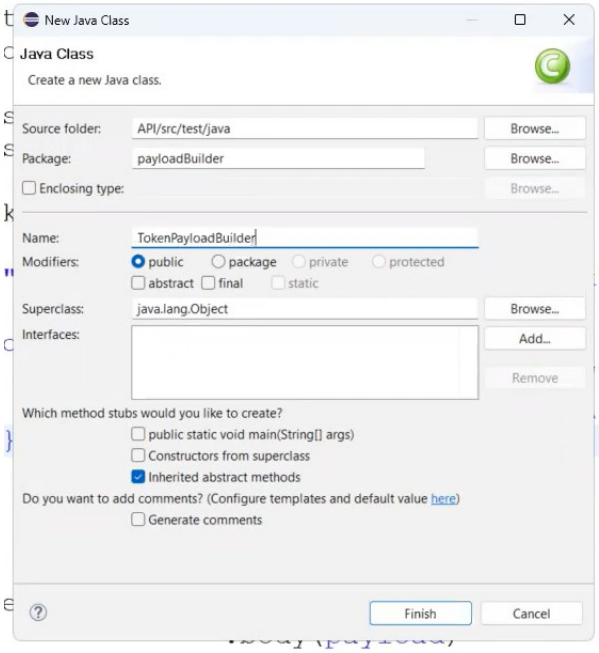
Result on Console:

```
{
    "result": {
```

```
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiI5MmE1ZmYxNy0xYjMyLTQ0NTEtOGI4NS0wZGI5MmQ3MDExYWEiLCJ
pYXQiOjE3NDE0NTE1NjksIm5iZiI6MTc0MTQ1MTU2OSwiZXhwIjoxNzcyOTg3NTY5LCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.gZfY2zYFdR01n5oGe_TLx6yIuGzUCJj1qCefpjwPoqw",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CQvcIYuFxIoS7kg+YR6rfAXX0BfteohQNihi
9FxfuflLMriZMS6FjAsHyKLrUNsJq+3pvJjB/MnGWn8pgCisUrGh2b+P3hYFJHQENp1rXAB/
Ip0bpbsPquUSS1pg3WjEEbditiJAoS8Q5PTKnG4cHPOMgbhA1juNz9eKSSngiGYs3XUttO1Tr16xpxcArtt/
Z2B1wNgfFJI7xZlY2N45/NENS0rmejhGX4lQm8yBp4JMuix16TVQmMPm/ZwUh8yEvVIhNtmaVjTUk63AJlQ/
M7NafO0ZPhANWObqpTBB0IiF",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```

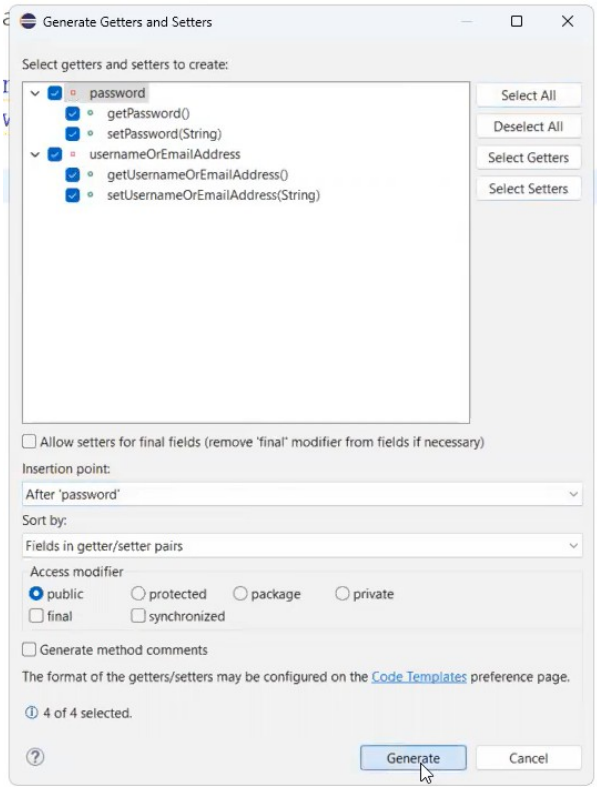
Second Method by using the encapsulation method to get the Access token:

Create a Java Class TokenPayLoadbuilder under payloadBuilder Package



Create getters and setters for the two variables on TokenPayLoadbuilder Java Class:

```
private String usernameOrEmailAddress;
private String password;
```




```java
package payloadBuilder;

public class TokenPayLoadbuilder {
   private String usernameOrEmailAddress;
   private String password;

   public String getUsernameOrEmailAddress() {
         return usernameOrEmailAddress;
   }
   public void setUsernameOrEmailAddress(String usernameOrEmailAddress) {
         this.usernameOrEmailAddress = usernameOrEmailAddress;
   }
   public String getPassword() {
         return password;
   }
   public void setPassword(String password) {
         this.password = password;
   }

}
```

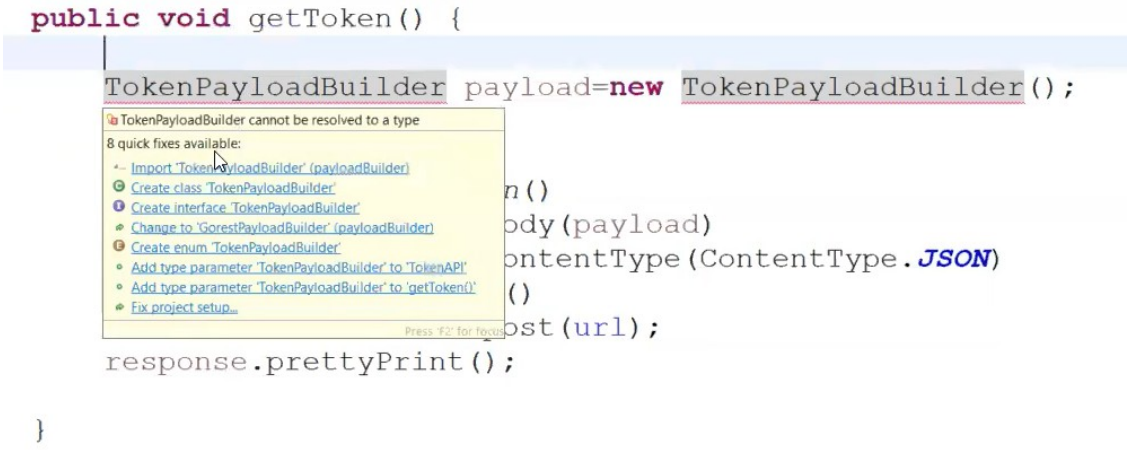
- **Create getToken() Method inside TokenAPI Class.**
- **Instantiate the Payload Object:**

Create `TokenPayLoadbuilder` Object that has the same name of the previous `TokenPayLoadbuilder` Java Class inside getToken() Method:

```
TokenPayLoadbuilder payload = new TokenPayLoadbuilder();
```

- **Import the Library for the Object,**
- **Add the Setters: Set the UsernameOrEmail Address**, and **the Password**:

```
payload.setUsernameOrEmailAddress("ElshanRasul");
payload.setPassword("Elshan123");
```




```
package mealB;

import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.TokenPayLoadbuilder;


public class TokenAPI {
	String url = "http://dev-mb.yoll.io/api/tokenauth/authenticate";

	@Test
	public void getToken() throws JsonProcessingException {

// create payload object
		TokenPayLoadbuilder payload = new TokenPayLoadbuilder();
```

```
            payload.setUsernameOrEmailAddress("ElshanRasul");
            payload.setPassword("Elshan123");

// Convert payload object to string and print it
            ObjectMapper objMap = new ObjectMapper();
            String strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(payload);
            System.out.println(strPayload);

// Submit POST request to get Token
     Response response = given()
                           .body(payload)
                           .contentType(ContentType.JSON)
                         .when()
                            .post(url);
            response.prettyPrint();
     }
}
```

RESULT:


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiJhNTRiNWUyNS1kOGE4LTRlMmMtYTM1Mi02MmZlODEyMjM3OTIiLCJ
pYXQiOjE3MTkwNzI0NzgsIm5iZiI6MTcxOTA3MjQ3OCwiZXhwIjoxNzUwNjA4NDc4LCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0._9qQcHFiHIXH8iuE_vP2s9hKVzlDzb4SPoaxV9KE2rw",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CRAl20FSluQ9Wg7s6Joq/
0gyZ0+8qRBwPTPpSAuHsdgm1QdBQyGYdYQmBIHrbjvwcmigYnRZmWhJFZt0bB8qL2h8wYhyxPa751rYpuJom8jF
ZOF2Dt/FXnEn/qtumy2WBSs9gfjbHAZCJVbxfWXdzcIWYK7c1hZq7QbvlLylR5/
F4GKNJOhniWLEXuD+0WEFQxxWJm1kjGXRXKMJQnsz8J8k7h+oKWrZENtwEPFUnyjQ8xcCThh570XCi7W4uN1f4+
fiOzARRkeQunYTE2TpiZWVvodZvC7xcpFgpjYrcszT",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
```

```
            "name": "Elshan",
            "surname": "Rasulras",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "55555555555",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": null,
            "city": "Fairfax",
            "postalCode": "22033",
            "tenantName": "",
            "accountType": 30,
            "birthDate": null,
            "businessType": null,
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": null,
            "countryStateId": null
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


Payload Object Explanation :

Creating Payload Object

The payload object in this context refers to the data that you need to send to the API in your POST request. In this example, it's an instance of a class named `TokenPayLoadbuilder`, which presumably has fields for the username or email address and password. This class is responsible for constructing the JSON payload that will be sent in the body of the POST request.

*Steps and **Details**:*

1. **Instantiate the Payload Object**:

   ```
   TokenPayLoadbuilder payload = new TokenPayLoadbuilder();
   ```

   This line creates a new instance of the `TokenPayLoadbuilder` class.

2. **Set the Username or Email Address**:

   ```
   payload.setUsernameOrEmailAddress("ElshanRasul");
   ```

   This line calls a method `setUsernameOrEmailAddress` on the `payload` object, passing the string `"ElshanRasul"` as the argument. This method sets the value of the username or email address in the payload.

3. **Set the Password**:

   ```
   payload.setPassword("Elshan123");
   ```

   Similarly, this line calls a method `setPassword` on the `payload` object, passing the string `"Elshan123"` as the argument. This method sets the value of the password in the payload.

MealB Homework : First Approach

Create a gift Expense from Expenses Tag

Create Token API request data from Java object.
Submit POST request to Token API.
Validate status code is 200.
Extract "accessToken" from response.

Create request payload from Java object for Create Gift Expense API.
Submit POST request to Create Gift Expense API (don't forget to add "accessToken" in header).
Validate status code is 200.
Validate if giftRecipient value in response is matching giftRecipient value in request payload.

We need to create 3 classes, Two for Getters and Setters.

CreateGiftExpense,

TokenPayloadBuilder, **GiftExpensePayloadBuilder**

URL That BE MUST BE USED DURING REQUEST IS:

http://dev-mb.yoll.io/api/expenses/create

Create : public void createExpense() method, and put under it : payload, validation, post requests

Look Below to the close illustrations with pictures of MealB Website of how to create a gift expenses that must be implemented on the Java Classes. They follows a proper API automation approach using Rest Assured and Jackson ObjectMapper.

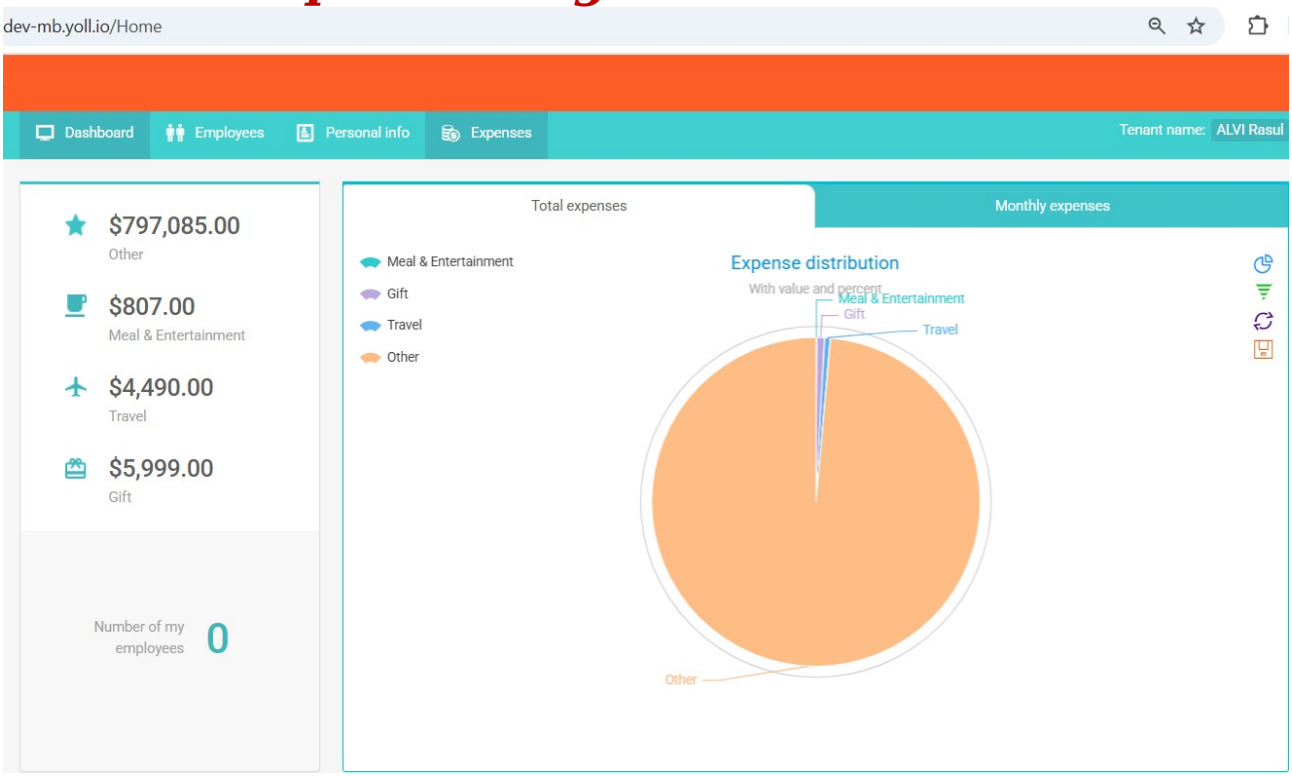
Login to MealB Website:

http://dev-mb.yoll.io/

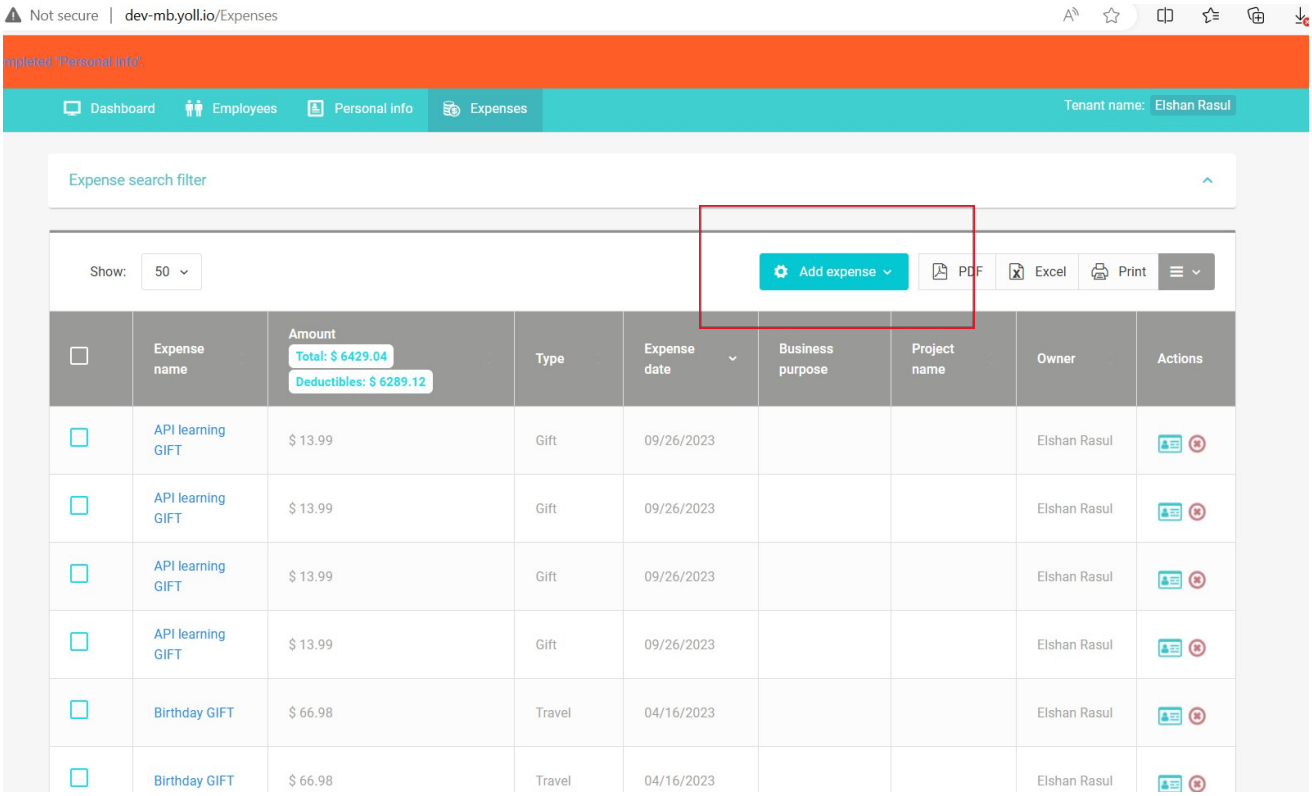
LOGIN :ElshanRasul

PW : Elshan123

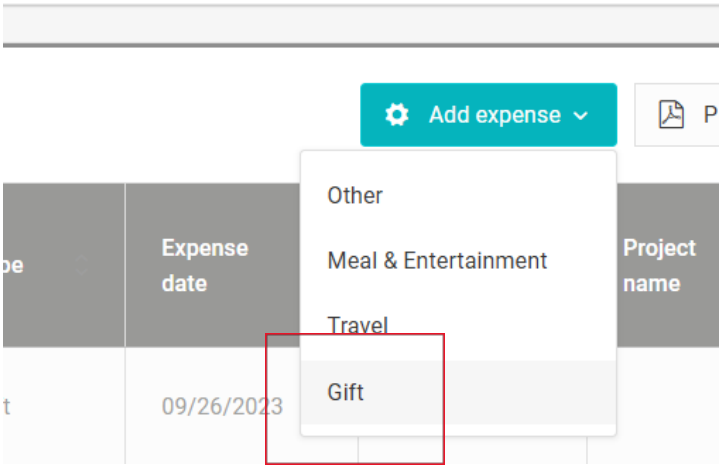
Click on Expenses Tag:

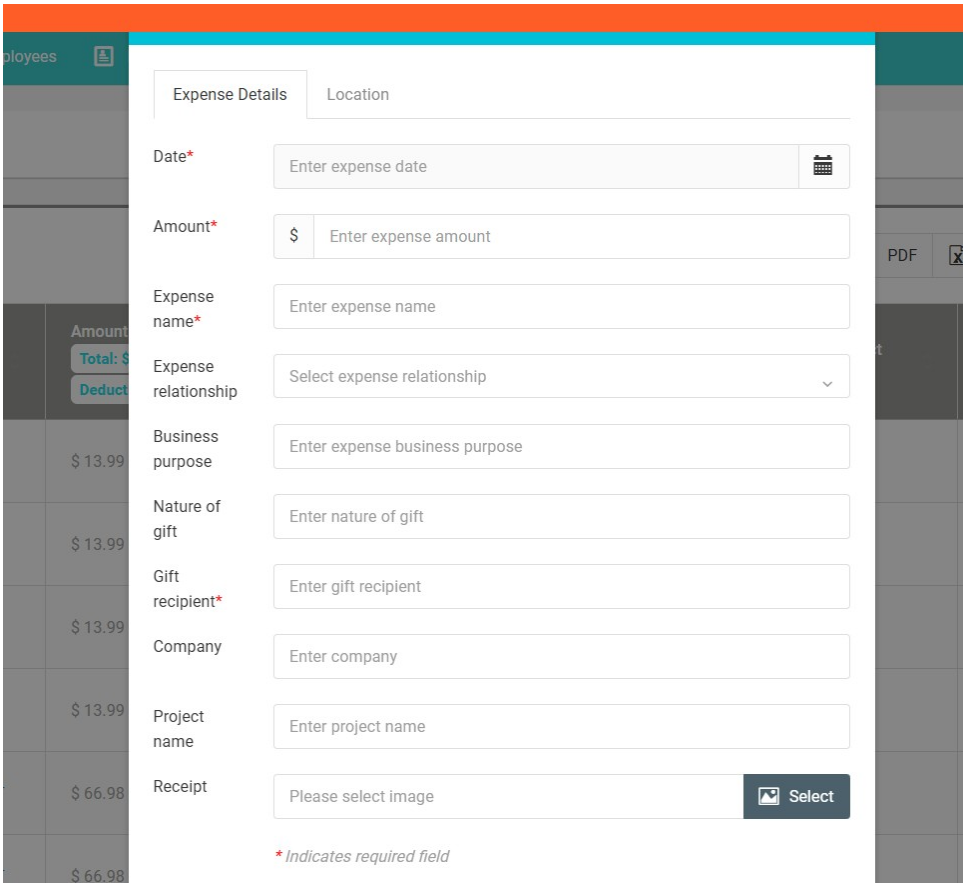


Click on Add expense:


Not secure | dev-mb.yoll.io/Expenses
Dashboard
Employees
Personal info
Expenses
Tenant name: Elshan Rasul
Expense search filter
Show: 50
Add expense
PDF
Excel
Print
Expense name
Amount
Total: $ 6429.04
Deductibles: $ 6289.12
Type
Expense date
Business purpose
Project name
Owner
Actions
API learning GIFT $ 13.99 Gift 09/26/2023 Elshan Rasul
API learning GIFT $ 13.99 Gift 09/26/2023 Elshan Rasul
API learning GIFT $ 13.99 Gift 09/26/2023 Elshan Rasul
API learning GIFT $ 13.99 Gift 09/26/2023 Elshan Rasul
Birthday GIFT $ 66.98 Travel 04/16/2023 Elshan Rasul
Birthday GIFT $ 66.98 Travel 04/16/2023 Elshan Rasul

Choose Gift


Add expense
Other
Meal & Entertainment
Travel
Gift
Expense date
Project name
09/26/2023

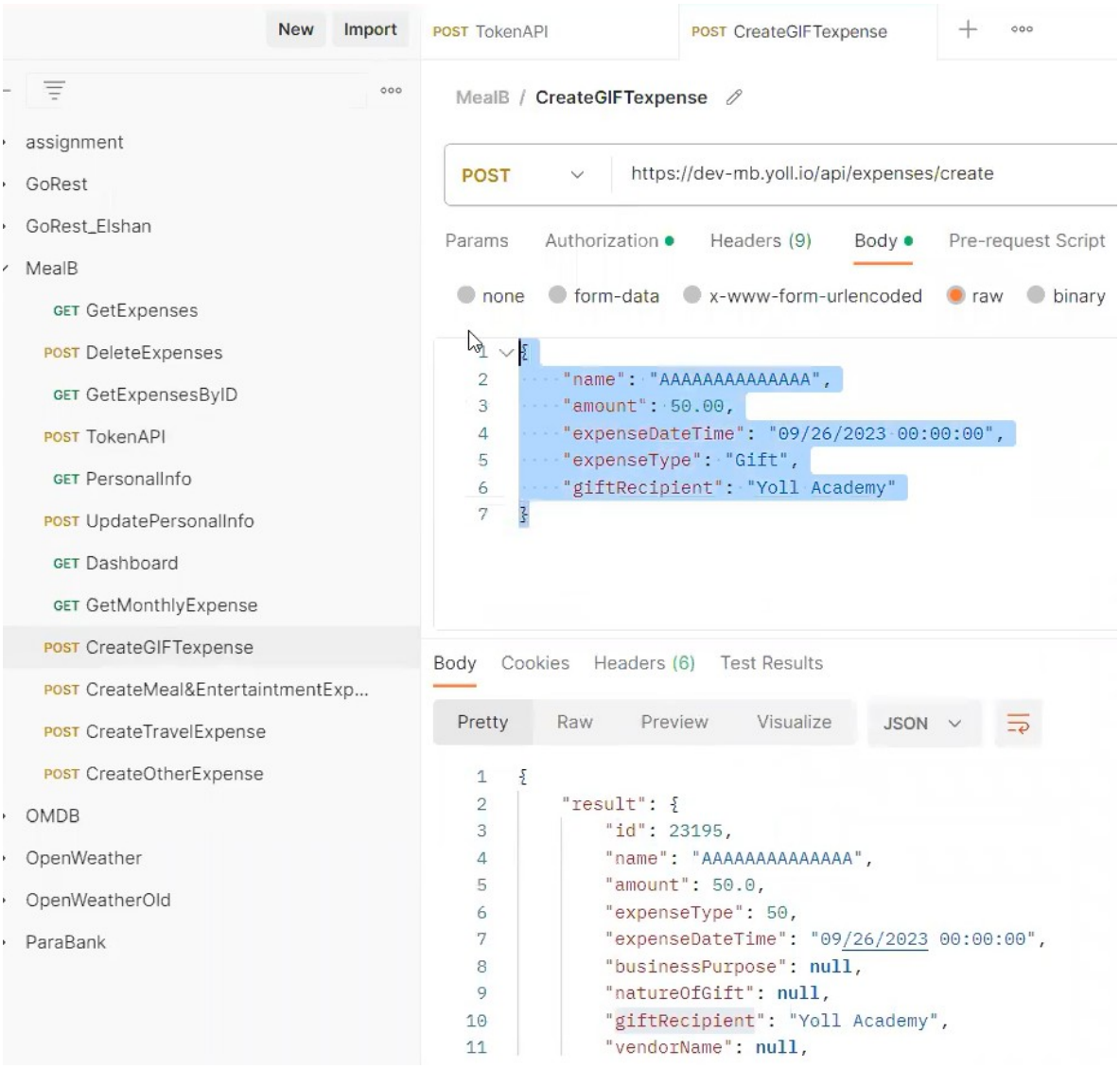


Java Class : TokenPayloadBuilder (Getters and setters)

```
package payloadBuilder;

public class TokenPayloadBuilder {
	private String usernameOrEmailAddress;
	private String password;

	public String getUsernameOrEmailAddress() {
		return usernameOrEmailAddress;
	}
	public void setUsernameOrEmailAddress(String
usernameOrEmailAddress) {
		this.usernameOrEmailAddress = usernameOrEmailAddress;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
```

}

Java Class : **GiftExpensePayloadBuilder** (Getters and setters)




```
package payloadBuilder;

public class GiftExpensePayloadBuilder {
	private String name;
	private double amount;
	private String expenseDateTime;
	private String expenseType;
	private String giftRecipient;
	public String getName() {
		return name;
```

```
	}
	public void setName(String name) {
		this.name = name;
	}
	public double getAmount() {
		return amount;
	}
	public void setAmount(double amount) {
		this.amount = amount;
	}
	public String getExpenseDateTime() {
		return expenseDateTime;
	}
	public void setExpenseDateTime(String expenseDateTime) {
		this.expenseDateTime = expenseDateTime;
	}
	public String getExpenseType() {
		return expenseType;
	}
	public void setExpenseType(String expenseType) {
		this.expenseType = expenseType;
	}
	public String getGiftRecipient() {
		return giftRecipient;
	}
	public void setGiftRecipient(String giftRecipient) {
		this.giftRecipient = giftRecipient;
	}


}
```

Java Class : CreateGiftExpense

```
package homework;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import org.junit.Test;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GiftExpensePayloadBuilder;
import payloadBuilder.TokenPayloadBuilder;

public class CreateGiftExpense {
    String tokenUrl =
"http://dev-mb.yoll.io/api/tokenauth/authenticate";
    String giftExpenseURL = "http://dev-mb.yoll.io/api/expenses/create";

    @Test
// createExpense() Method
     public void createExpense() throws
JsonProcessingException  {
        // Create Token API request data from Java object
        TokenPayloadBuilder payload = new TokenPayloadBuilder();
        payload.setUsernameOrEmailAddress("ElshanRasul");
        payload.setPassword("Elshan123");

        // Convert payload object to string and print it
        ObjectMapper objMap = new ObjectMapper();
        String strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(payload);
        System.out.println(strPayload);

        // Submit POST request to Token API
        Response response = given()
                .body(payload)
                .contentType(ContentType.JSON)
                .when()
                .post(tokenUrl);
        response.prettyPrint();
```

```
        // Validate status code is 200
        assertEquals(200, response.getStatusCode());

        // Extract "accessToken" from response
        String token = JsonPath.read(response.asString(),
"$.result.accessToken");
        System.out.println(token);

        // Create request payload from Java object to Create Gift
Expense API
        GiftExpensePayloadBuilder giftPayload = new
GiftExpensePayloadBuilder();
        giftPayload.setName("API learning GIFT");
        giftPayload.setAmount(13.99);
        giftPayload.setExpenseDateTime("09/26/2023 04:30:12");
        giftPayload.setExpenseType("Gift");
        giftPayload.setGiftRecipient("Cohort12 Students");

        // Convert payload object to string and print it
        String strGiftPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(giftPayload);
        System.out.println(strGiftPayload);

        // Submit POST request to Create Gift Expense API (don't forget
to add "accessToken" in header)
        Response giftResponse = given()
                .header("Authorization", "Bearer " + token)
                .body(giftPayload)
                .contentType(ContentType.JSON)
                .when()
                .post(giftExpenseURL);
        giftResponse.prettyPrint();

        // Validate status code is 200
        assertEquals(200, giftResponse.getStatusCode());

        // Validate if giftRecipient value in response is matching
giftRecipient value in request payload
        String responseRecipient =
JsonPath.read(giftResponse.asString(), "$.result.giftRecipient");
        assertEquals(giftPayload.getGiftRecipient(), responseRecipient);
    }
}
```

Why We Used a Getter (getGiftRecipient()) Instead of a Setter (setGiftRecipient()) in assertEquals() ?

1 - getGiftRecipient() Retrieves Data

- **getGiftRecipient() is a getter method** → It retrieves the stored value.
- We use it to **fetch the originally set value** so we can compare it with the API response.

```
String expectedRecipient = giftPayload.getGiftRecipient();
// Retrieves stored value
```

2 - setGiftRecipient() Modifies Data

1. **setGiftRecipient() is a setter method → It modifies the object's data.**
2. **We use the Setters to create payload from Java object**
3. If we used setGiftRecipient(), we would be **changing** the value instead of verifying it.

❌ Incorrect: Modifying Instead of Comparing

```
giftPayload.setGiftRecipient(responseRecipient);
// ❌ Changes value instead of retrieving
```

Result:


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
```

8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiIzMmJiMDI2OC1jOGQ5LTQzNGQtYWQwZC0zY2FhZmQ3OTE5MWEiLCJ
pYXQiOjE3MTkwOTY4MzgsIm5iZiI6MTcxOTA5NjgzOCwiZXhwIjoxNzUwNjMyODM4LCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.10lp82LHX07wfTuOYF2sYdg5oL8zPQMLg6RFGlqpSyE",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CcaLAl5cDUwnQwBXP+ABgEkqIJLRA8A0lkWR
fbNnzhHNLrpdkmuJzaCu8pWvu0KBFIGCdEU7+gbPa40ZxaGMcyvVQZbpgWarHHcH0zoSOGckcOJTd7gBmzcPbtq
oJpY08xSD6Qsy+aeaZGYJ5bGTYXLakM1hRKGAT2j5c8aXb7qiJJdue/
szh7JN7XaCStOxqgXDELxUQv+4SzyRamZnxRGTvYaub6/
aXfgbPaJrKhaM5u5UiVjdS8t+eNUOlRh9OTfu+RJIGQU1BGlxSD4c5mbte2T8gLiUpGiu6QqJeqLM",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasulras",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "55555555555",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": null,
            "city": "Fairfax",
            "postalCode": "22033",
            "tenantName": "",
            "accountType": 30,
            "birthDate": null,
            "businessType": null,
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": null,
            "countryStateId": null
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1Lz
A1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29hc
C5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW50
aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi8
vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wbG

95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQiOiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiIzMmJiMDI2OC1jOGQ5LTQzNGQtYWQwZC0zY2FhZmQ3OTE5MWEiLCJpYXQiOjE3MTkwOTY4MzgsIm5iZiI6MTcxOTA5NjgzOCwiZXhwIjoxNzUwNjMyODM4LCJpc3MiOiJNZWFsQiIsImF1ZCI6Ik1lYWxCIn0.10lp82LHX07wfTuOYF2sYdg5oL8zPQMLg6RFGlqpSyE


```
{
  "name" : "API learning GIFT",
  "amount" : 13.99,
  "expenseDateTime" : "09/26/2023 04:30:12",
  "expenseType" : "Gift",
  "giftRecipient" : "Cohort12 Students"
}
{
    "result": {
        "id": 27625,
        "name": "API learning GIFT",
        "amount": 13.99,
        "expenseType": 50,
        "expenseDateTime": "09/26/2023 04:30:12",
        "businessPurpose": null,
        "natureOfGift": null,
        "giftRecipient": "Cohort12 Students",
        "vendorName": null,
        "destinationOfTravel": null,
        "company": null,
        "projectName": null,
        "longitude": null,
        "latitude": null,
        "placeName": null,
        "placeAddress": null,
        "placeId": null,
        "placeIcon": null,
        "expenseRelationshipId": null,
        "travelExpenseTypeId": null,
        "otherExpenseNameId": null,
        "otherRelationship": null,
        "receiptFile": null,
        "receiptUrl": null
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


Explaination :

`package homework;`

 Defines the package where the class is located (`homework`).

```
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
```

 Static imports allow using `given()` and `assertEquals()` without prefixing them with class names.

```
import org.junit.Test;
```

 Imports JUnit to define test cases.

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;
```

 Jackson ObjectMapper → Used for JSON serialization and deserialization.
 JsonPath → Extracts values from API responses.

```
import io.restassured.http.ContentType;
import io.restassured.response.Response;
```

 Rest Assured classes → Handles API requests and responses.

```
import payloadBuilder.GiftExpensePayloadBuilder;
import payloadBuilder.TokenPayloadBuilder;
```

 Custom classes → Used to build request payloads for authentication and expense creation.

Step 1: Define API URLs

```
String tokenUrl =
"http://dev-mb.yoll.io/api/tokenauth/authenticate";
String giftExpenseURL =
"http://dev-mb.yoll.io/api/expenses/create";
```

 Defines **API endpoints** for token authentication and expense creation.

Step 2: Create Authentication Token

```
@Test
public void createExpense() throws JsonProcessingException  {
```

 JUnit test method to create an expense.
 Throws `JsonProcessingException` → In case JSON serialization fails.

```
// Create Token API request data from Java object
TokenPayloadBuilder payload = new TokenPayloadBuilder();
payload.setUsernameOrEmailAddress("ElshanRasul");
payload.setPassword("Elshan123");
```

 Creates a `TokenPayloadBuilder` object with username and password.

```
// Convert payload object to JSON string and print it
ObjectMapper objMap = new ObjectMapper();
String strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(pay
load);
System.out.println(strPayload);
```

 Serializes Java object (payload) to JSON string using `ObjectMapper`.
 `writeValueAsString(payload)` → Converts Java object to **JSON String**.
 `writerWithDefaultPrettyPrinter()` → Formats JSON output for readability.

```
// Submit POST request to Token API
Response response = given()
    .body(payload)  // Sends JSON payload
    .contentType(ContentType.JSON)  // Sets content type as
JSON
    .when()
    .post(tokenUrl);
response.prettyPrint();
```

 Sends POST request to `tokenUrl` to authenticate and get a token.
 Response is printed for debugging.

```
// Validate status code is 200
assertEquals(200, response.getStatusCode());
```

 Assertion to ensure successful response.

```
// Extract "accessToken" from response
String token = JsonPath.read(response.asString(),
"$.result.accessToken");
System.out.println(token);
```

 Extracts accessToken from JSON response using **JsonPath**.
 Prints the token to the console.

Step 3: Create Gift Expense

```
// Create request payload from Java object to Create Gift
Expense API
GiftExpensePayloadBuilder giftPayload = new
GiftExpensePayloadBuilder();
giftPayload.setName("API learning GIFT");
giftPayload.setAmount(13.99);
giftPayload.setExpenseDateTime("09/26/2023 04:30:12");
giftPayload.setExpenseType("Gift");
giftPayload.setGiftRecipient("Cohort12 Students");
```

 Creates request payload for **Gift Expense API**.
 Populates required fields using setters.

```
// Convert payload object to JSON string and print it
String strGiftPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(gif
tPayload);
System.out.println(strGiftPayload);
```

 Serializes giftPayload to JSON string.

```
// Submit POST request to Create Gift Expense API (don't
forget to add "accessToken" in header)
Response giftResponse = given()
    .header("Authorization", "Bearer " + token)  // Adds
authentication token
    .body(giftPayload)  // Sends gift expense data
    .contentType(ContentType.JSON)
    .when()
    .post(giftExpenseURL);
```

```
giftResponse.prettyPrint();
```

 Sends authenticated POST request to create a gift expense.
 Authorization token is passed in the header.

```
// Validate status code is 200
assertEquals(200, giftResponse.getStatusCode());
```

 Asserts that expense creation was successful.

```
// Validate if giftRecipient value in response is matching
giftRecipient value in request payload
String responseRecipient =
JsonPath.read(giftResponse.asString(),
"$.result.giftRecipient");
assertEquals(giftPayload.getGiftRecipient(),
responseRecipient);
```

 Extracts `giftRecipient` from API response.
 Compares with request payload to ensure the correct data was stored.

MealB Homework : Second Approach BY Calling METHODS

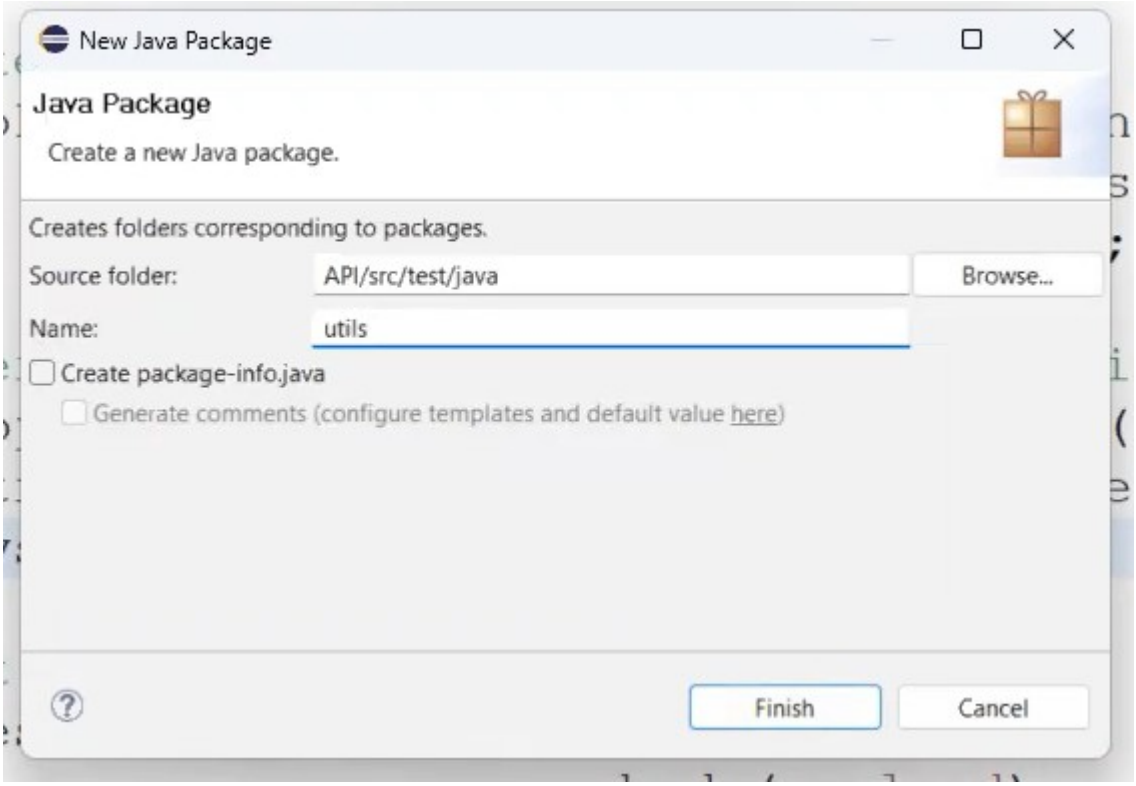
Some codes are uesd many times, we can create create methods, and call them when we need them without writing the same codes many times.

Keep the two classes : TokenPayloadBuilder and `GiftExpensePayloadBuilder` that contain Setters and Getters.

Put the methods inside `FunctionLibrary Java class instead of` CreateGiftExpense class.

1- Create a package under src/test/java, name it: utils.

2- Create under utils package a class that will contains the reusable functions and methods, name it: `FunctionLibrary`.



3- Create the following methods:

- **getToken() → Sends login credentials, retrieves authentication token.**
- **createTokenPayload() → Creates request body with username & password.**
- **convertObjToString(Object obj) → Converts Java object → JSON.**

4- Create the Class : GetExpenses extended from FunctionLibrary, and has:

- **getExpenseCount () → to get the result of the expenses.**

Key Differences between a return method and a void method:

Feature	Return Method (`returnType ≠ void`)	Void Method (`void`)
Returns a value?	Yes (`return statement`)	No

Feature	Return Method (returnType ≠ void)	Void Method (void)
Can store result?	✅ Yes (`int result = method();`)	❌ No (`void` means nothing to store)
Can be used in expressions?	✅ Yes (`x = method() + 5;`)	❌ No
Use case	Calculations, fetching values	Printing, updating variables

class : FunctionLibrary

```
package utils;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.TokenPayloadBuilder;

public class FunctionLibrary {
    String tokenUrl =
"http://dev-mb.yoll.io/api/tokenauth/authenticate";

    /**
     * @author ElshanRasulov This method submits POST request
to create token and
     *         retrieve token from response and return
     */

// getToken() :Is a Method to Get API Token, and returns a
String (the token value).
    public String getToken() {
        Response response = given()
// Calls method to get payload → Generates request data
(username & password).
.body(createTokenPayload())
.contentType(ContentType.JSON)
.when()
.post(tokenUrl);
        response.prettyPrint();
```

```
        assertEquals(200, response.getStatusCode());

//extract "accessToken" from response
        String token = JsonPath.read(response.asString(),
"$.result.accessToken");
        return token;
    }

    /**
     * @author ElshanRasulov This method create token request
payload data and
     *         convert it to string and return
     */
//This method creates the request body (username/password) for
token authentication.

    public String createTokenPayload() {
//Create Token Request Object
        TokenPayloadBuilder payload = new
TokenPayloadBuilder();
        payload.setUsernameOrEmailAddress("ElshanRasul");
        payload.setPassword("Elshan123");
//Calls convertObjToString(payload) to convert the request
object into JSON.
        return convertObjToString(payload);
    }

    /**
     * @author ElshanRasulov This method convert payload
objects to String format
     */

// Converts any Java object into a JSON-formatted string.
    public String convertObjToString(Object obj) {

//Initialize JSON String
        String strPayload = null;
        try {
            ObjectMapper objMap = new ObjectMapper();
            strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(obj
);
            System.out.println(strPayload);
```

```
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
//Returns the JSON string representation of the object.
        return strPayload;
    }
}
```

Class : GetExpenses extends FunctionLibrary

```
package mealB;

import java.util.List;

import org.junit.Test;

import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;

import io.restassured.response.Response;
import utils.FunctionLibrary;

//Remove the letter S from HTTPS. Otherwise, it will be an
error.
public class GetExpenses extends FunctionLibrary {
    String getExpenseURL =
"http://dev-mb.yoll.io/api/Expenses";

    @Test
    public void getExpenseCount() {
// get token
        String token = getToken();
        System.out.println(token);
// submit GET request for expenses
        Response repsonse = given().header("Authorization",
"Bearer " + token).when().get(getExpenseURL);
        repsonse.prettyPrint();
//get count of expense

//extracts a list of expenses from a JSON response using
JsonPath.
```

```
        List<String> expenses =
JsonPath.read(repsonse.asString(), "$.result[*]");
//Retrieve the id:
List<String> id = JsonPath.read(repsonse.asString(),
"$.result[*].id");
        System.out.println(id);
//Print the total of expenses
        System.out.println(expenses.size());
    }
}
```

Look at `"result": [` to see the result

RESULT:


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiI2ODc1NWNjMC1jNTFkLTQ5NDAtYWViMS1mMjczYTI4NjgwN2YiLCJ
pYXQiOjE3MTkxNzgxMTAsIm5iZiI6MTcxOTE3ODExMCwiZXhwIjoxNzUwNzE0MTEwLCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.lcVnUbxZ68ACBzz-F1_68qAw0dXa2rzE0O49mB2bOnc",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CVnr92QjUaXoTQAMDvwSjj0oAP2QBqxRXDL1
Hj1TsmiQwbVR1cLdOwW89hY9S+M/
CMzV2kDzH+X4sCsNhsLkXcQGJIETHkyXahk+6o+RHrKSgWYZFg7+uT3kgWIYaeyS4zhSjBRBQDSBL/
SNNYfzj3BONStW30kKTBc7GONUv4efCqdpgrj6/
O9NICruNseGQMdd5qD13KxEQgIiK0WkgNUcp+dcWMHrlZ83Eff9TqfRbsVCeho7RrohBbU1P5kOYeVXxDBCGsIu
ddY6zQBKifKPuY+RUTtx+pr1VgvYF7mn",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasulras",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
```

```
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "55555555555",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": null,
            "city": "Fairfax",
            "postalCode": "22033",
            "tenantName": "",
            "accountType": 30,
            "birthDate": null,
            "businessType": null,
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": null,
            "countryStateId": null
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1Lz
A1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29hc
C5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW50
aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi8
vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wbG
95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQiO
iIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiI2ODc1NWNjMC1jNTFkLTQ5NDAtYWViMS1mMjczYTI4NjgwN2YiLCJp
YXQiOjE3MTkxNzgxMTAsIm5iZiI6MTcxOTE3ODExMCwiZXhwIjoxNzUwNzE0MTEwLCJpc3MiOiJNZWFsQiIsImF
1ZCI6Ik1lYWxCIn0.lcVnUbxZ68ACBzz-F1_68qAw0dXa2rzE0O49mB2bOnc
{
    Result of getExpenseCount()
      "result": [
        {
            "id": 27628,
            "name": "API learning GIFT",
            "amount": 13.99,
            "expenseType": 50,
            "expenseDateTime": "09/26/2023 04:30:12",
            "businessPurpose": null,
            "natureOfGift": null,
            "giftRecipient": "Cohort12 Students",
            "vendorName": null,
            "destinationOfTravel": null,
            "company": null,
            "projectName": null,
            "longitude": null,
            "latitude": null,
            "placeName": null,
            "placeAddress": null,
            "placeId": null,
            "placeIcon": null,
            "expenseRelationshipId": null,
```

```
            "travelExpenseTypeId": null,
            "otherExpenseNameId": null,
            "otherRelationship": null,
            "receiptFile": null,
            "receiptUrl": null
        },
        {***},{***},{***},{***},{***},{***},{***},{***},{***},{***},{***},{***},
{***}, ,{***},{***},{***},{***},{***},{***},{***},{***},{***},{***},{***},{***},
{***},…….
```


.
.
. **99 RESULTS**
.
.
.


```
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


100

***Retrieve the id:

Result:

[27628,27627,27626,27625,27624,27583,27582,27581,27580,27579,27578,27577,27576,27575,27574,27573,27572,27571,27570,27569,27568,27567,27566,27565,27564,27563,27562,27561,27560,27559,27558,27557,27556,27555,27554,27553,27552,27551,27550,27549,27548,27547,27546,27545,27544,27543,27542,27541,27540,27539,27538,27537,27536,27535,27534]

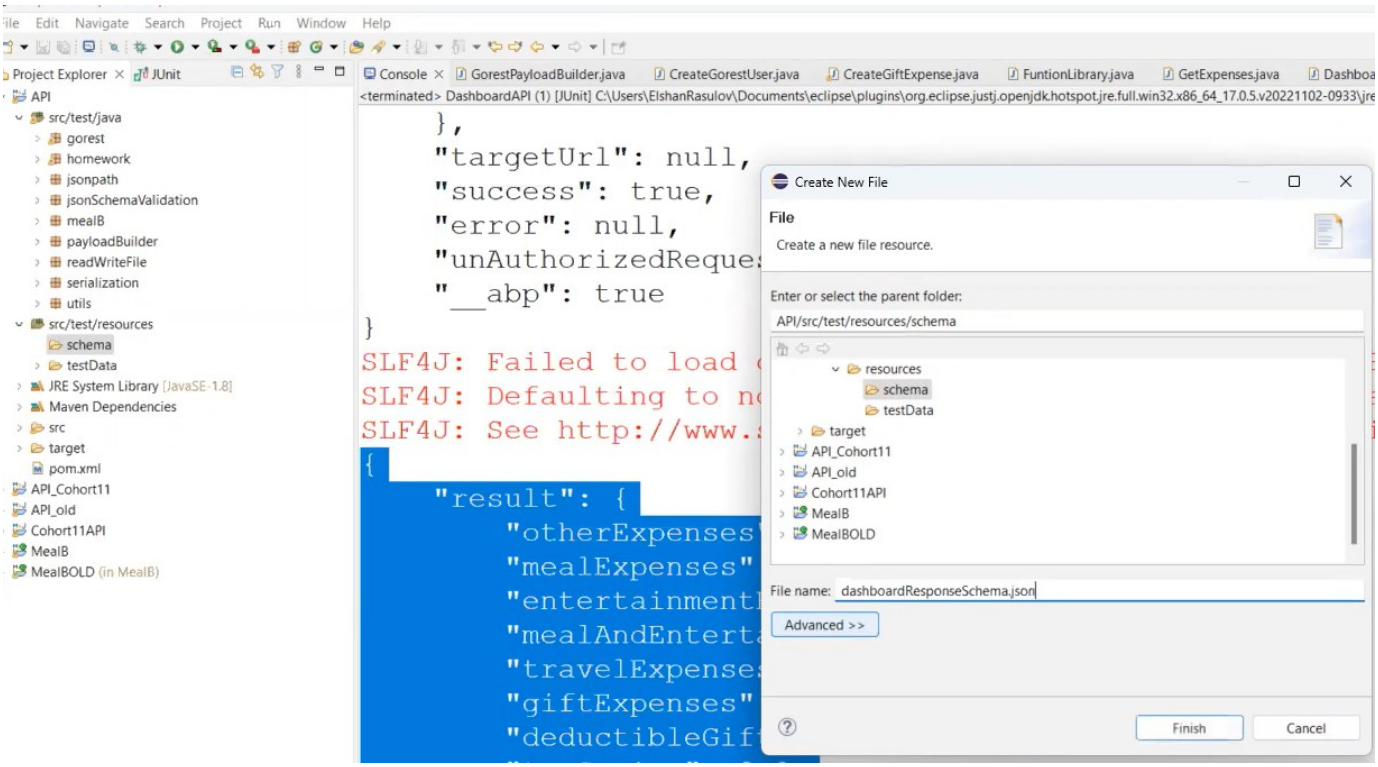
JSON schema validation

JSON Schema validation ensures that the response from an API matches the expected structure, data types, and required fields.

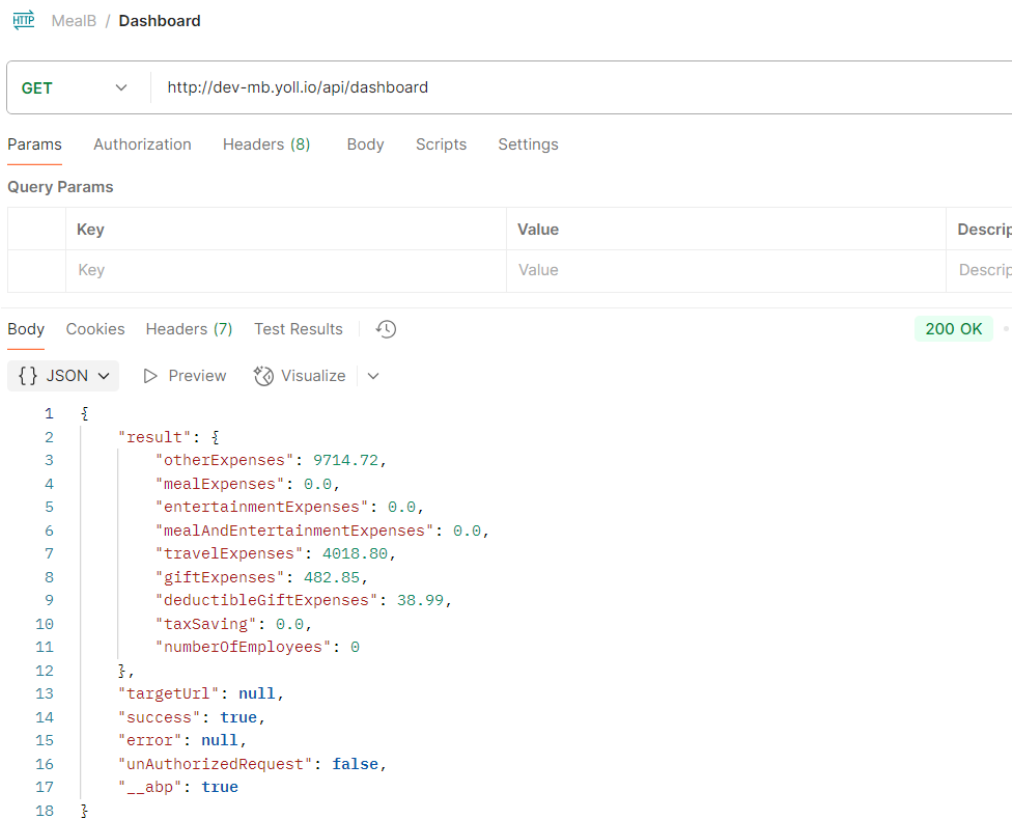
1. It needs: **JSON schema validator Library** From: https://mvnrepository.com/
2. Copy/paste on **pom.xml**:

```
<!--
        https://mvnrepository.com/artifact/com.github.java-
json-tools/json-schema-validator -->
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>json-schema-validator</artifactId>
            <version>5.3.0</version>
        </dependency>
```

3. **Create a package, name it : jsonSchemaValidation.**
4. **Create a folder (schema) under *src/test/resources***
5. **Create a JSON file under schema: dashboardResponseSchemachema.json**

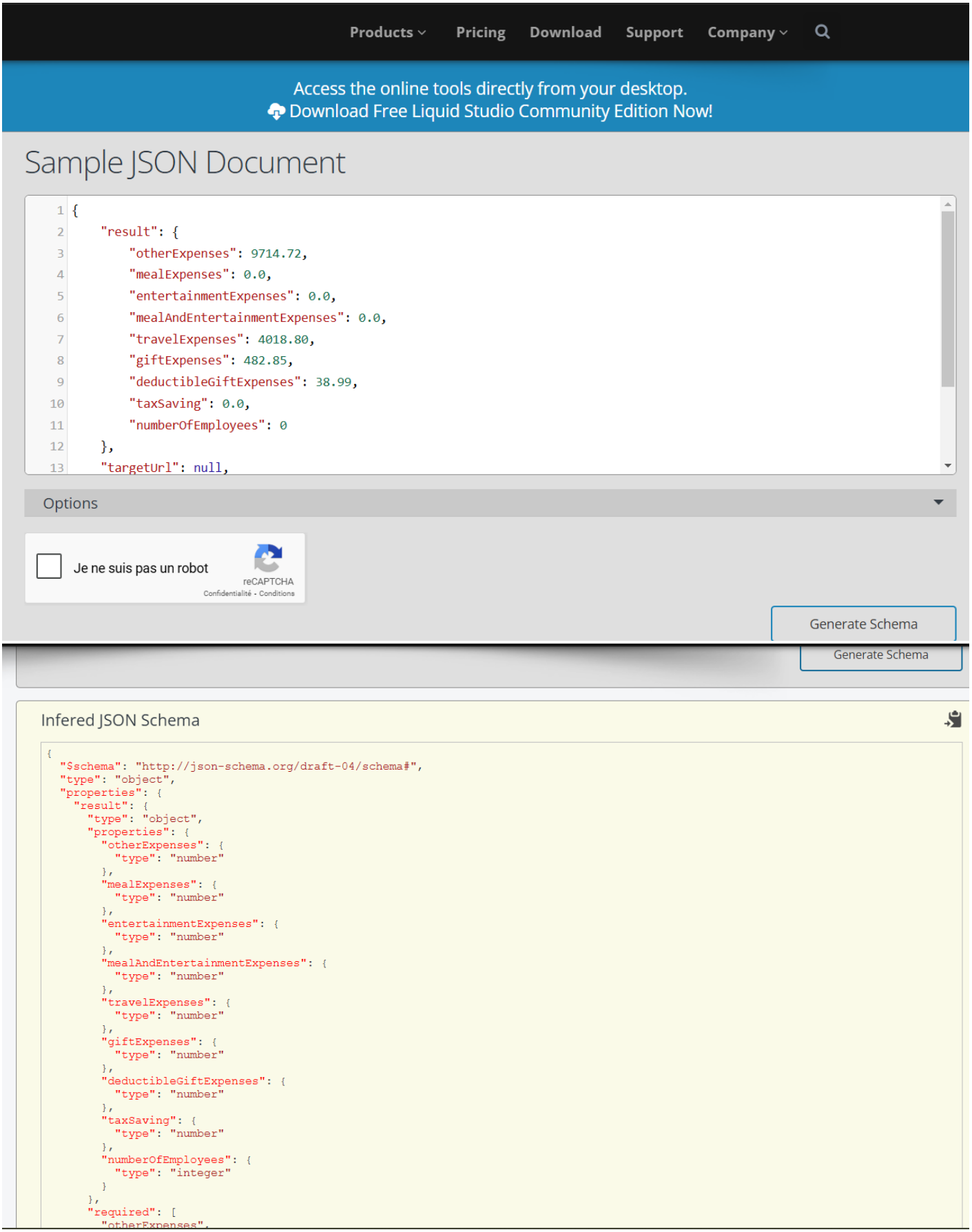


6. **Go to Postman, Copy Json Data from Get Method of the Dashboard:**



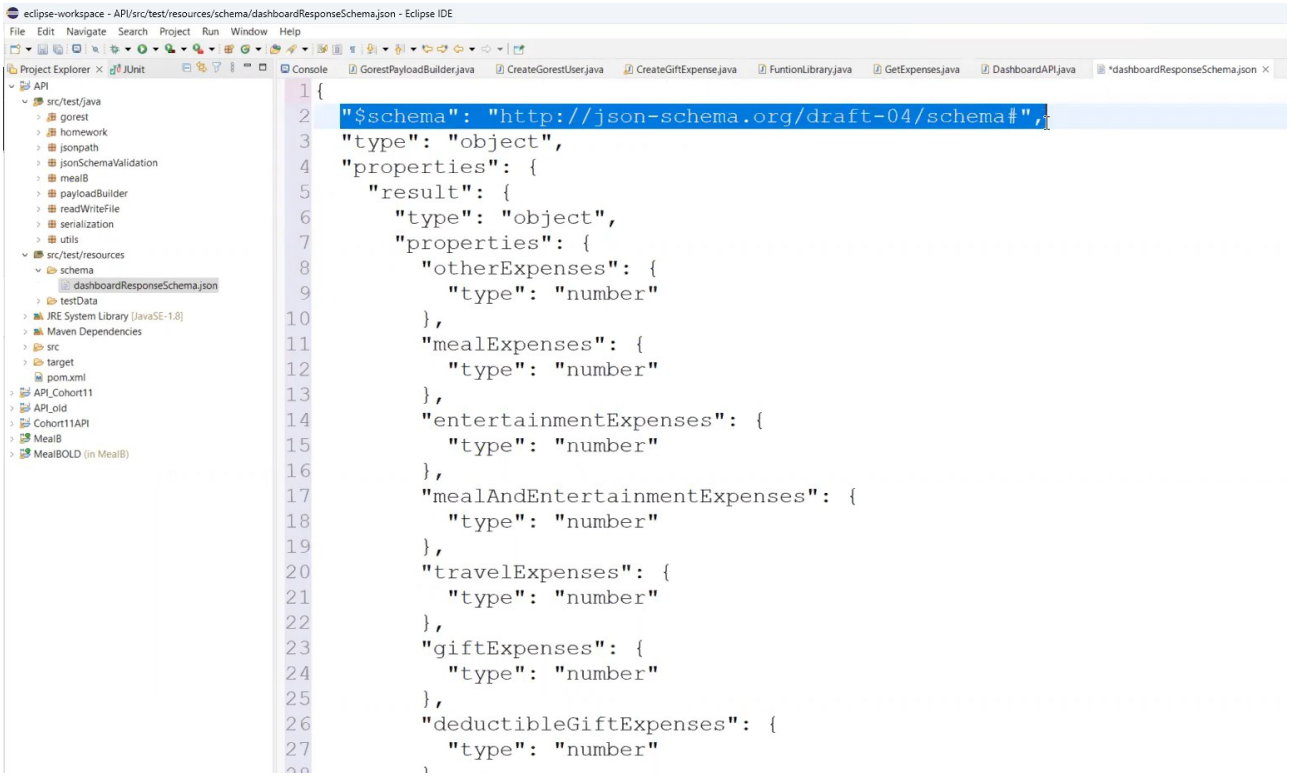

```
{
    "result": {
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


7. **Convert JSON Data to JSON Schema:**
   Paste Json Data of Dashboard on the Generator



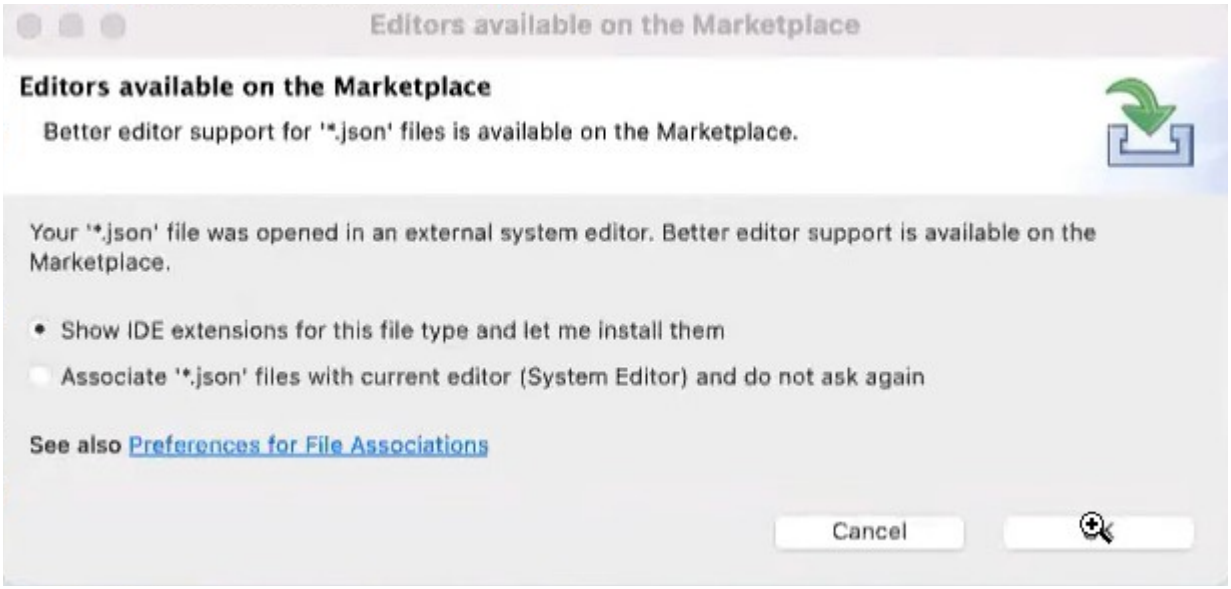
8. **Paste the JSON Schema on the JSON file: dashboardResponseSchemachema.json**

9. Delete the first line.



If you have this message, choose the second option:

Associate '*.json' files with current editor (System Editor) and do not ask again


Editors available on the Marketplace
Editors available on the Marketplace
Better editor support for '*.json' files is available on the Marketplace.
Your '*.json' file was opened in an external system editor. Better editor support is available on the Marketplace.
Show IDE extensions for this file type and let me install them
Associate '*.json' files with current editor (System Editor) and do not ask again
See also Preferences for File Associations
Cancel

10. Put the Path of json file : schema/dashboardResponseSchema.json

```
.body(matchesJsonSchemaInClasspath("schema/dashboardResponseSchema.json"));
```

Do not forget to import :

```
import static
io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspath;
```

Validate Response with JSON Schema

```
repsonse.then().assertThat()

.body(matchesJsonSchemaInClasspath("schema/dashboardResponseSchema.json"));
```

Schema Validation Process:

1. **matchesJsonSchemaInClasspath("schema/dashboardResponseSchema.json")**:
   - Validates the API response against the JSON schema file.
   - The schema is located at **src/test/resources/schema/dashboardResponseSchema.json**.
2. If the response **doesn't match the schema**, the test **fails** with an assertion error.

DashboardAPI Class

```
package jsonSchemaValidation;

import static io.restassured.RestAssured.given;
import static
io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspath;
import org.junit.Test;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class DashboardAPI extends FunctionLibrary {
  String url = "http://dev-mb.yoll.io/api/dashboard";

  @Test
  public void testJsonSchema() {
      String token = getToken();
// submit GET request for expenses
```

```
        Response repsonse = given().header("Authorization",
    "Bearer " + token).when().get(url);
        repsonse.prettyPrint();
    //validate json schema
        repsonse.then().assertThat()
    .body(matchesJsonSchemaInClasspath("schema/
    dashboardResponseSchema.json"));
      }
    }
```

Result : ..

 Passes validation because:

- **All required fields are present**.
- **Each field has the correct data type**.


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiJhYTUzNmYyMy05OGFlLTRkYTQtYmM2Mi1lZWIxYzgxMTc4NGYiLCJ
pYXQiOjE3NDE1NTY4NTQsIm5iZiI6MTc0MTU1Njg1NCwiZXhwIjoxNzczMDkyODU0LCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.RAOMkviNwk9sIh9sNuYhtxqfhmSWHk5LYICMejCi1aU",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CSzkQUS+DfN1y9vtdzPpL8rGUDuVvgFaP4Jb
tCvU8J+RW/0dPr7+VEFY7sdpI0CaViPNSaNcH3bAiRWfIRl00fUldX0PyyVsehv4YjHiieu5YnPYEP/
hqDQGuVa9IMwZQPooJg+3DwGhGHTXtrtD5spY6HaxNYDpJdC8IIBT9dyHEIQU6Sm0Uwcyaz0v68QQdnHmWtfmt2
MQZ90GREVmWu7VleZ6+HMs74pjlKBUz1h24zGfPpnMeO2fcUEoY90BHn0sTIclHesXHY5sDuxcToZTFkGqWaZu6
5U9499WJj97",
        "expireInSeconds": 31536000,
        "userId": 10781,
```

```
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


❌ Example of an Invalid API Response


```
{
  "result": {
    "otherExpenses": 500.75,
    "mealExpenses": "200.50",  // ❌ ERROR: Should be a number, but it's a string
```

```
    "entertainmentExpenses": 150.00,
    "mealAndEntertainmentExpenses": 350.50,
    "travelExpenses": 1000.00,
    "giftExpenses": 300.00,
    "deductibleGiftExpenses": 150.00,
    "taxSaving": 250.00
    // ❌ ERROR: "numberOfEmployees" is missing
  },
  "targetUrl": null,
  "success": true,
  "error": null,
  "unAuthorizedRequest": false,
  "__abp": true
}
```

❌ Fails validation because:

1. `mealExpenses` **should be a number** but is a **string**.
2. `numberOfEmployees` **is missing** (required field).

✅ JSON Schema Validation in Java (JUnit & Rest Assured)

To validate this schema in **JUnit** using **Rest Assured**, use:

```java
@Test
public void validateDashboardResponseSchema() {
    String token = getToken(); // Retrieve authentication token

    // Send GET request to dashboard API
    Response response = given()
        .header("Authorization", "Bearer " + token)
        .when()
        .get("http://dev-mb.yoll.io/api/dashboard");

    response.prettyPrint(); // Print response for debugging

    // Validate response against JSON Schema
    response.then().assertThat()
        .body(matchesJsonSchemaInClasspath("schema/dashboardResponseSchema.json"));
}
```

✅ If the response structure matches the schema, the test passes.
❌ If it doesn't match, the test fails with an error message.

How to Convert JSON Language to JSON Schema ?

We need to use a JSON schema Generator

Use this Website: [Free Online JSON to JSON Schema Converter (liquid-technologies.com)]


```
{
"name": "John An Smith",
"email": "api11@gmail.com",
"gender": "male",
"status": "active"
}
```


Infered JSON Schema


```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "email": {
      "type": "string"
    },
    "gender": {
      "type": "string"
    },
    "status": {
      "type": "string"
    }
  },
  "required": [
    "name",
    "email",
    "gender",
    "status"
  ]
}
```

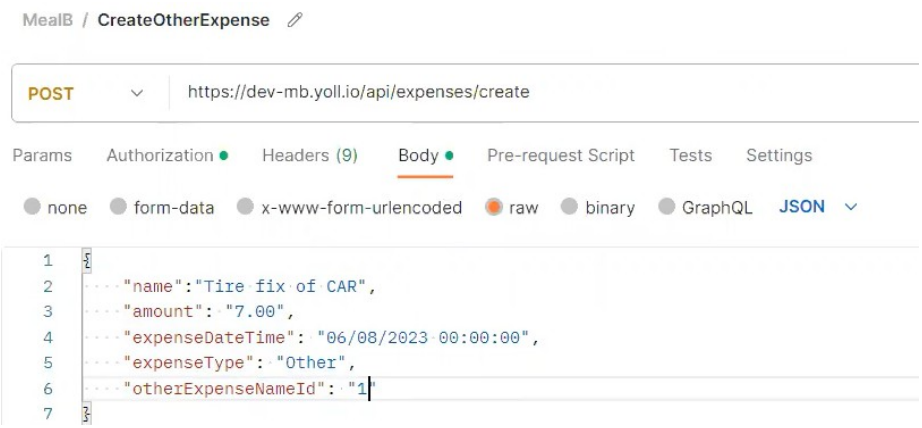
TASK : Create Other Expense

Login into MealB. Click on Expenses Tag. Add Expenses.
Choose Expenses Name: 1: *Electricity*, 2: *Rent*, 3: *Gas*, 4: *Other*

On the Post Method named : CreateOtherExpense, *When we put : number 1 on* OtherExpenseNameID field Then SEND it, The name on the result must be changed to : Electricity, and so on with the other numbers:2, 3.

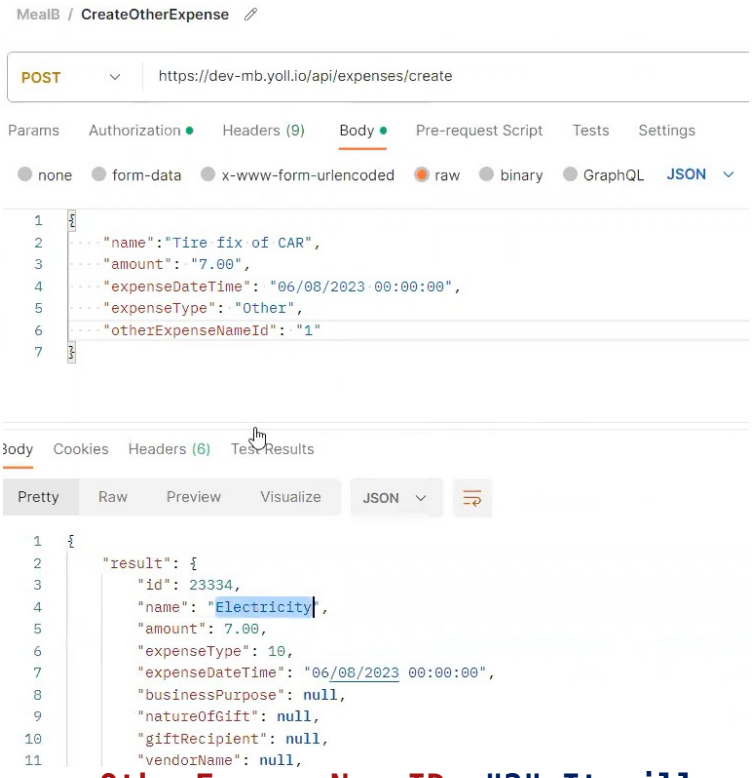
2 To Rent, 3 To Gas. And Whatever name we pass for number 4, it must show the same name on Body (Raw):Request to the Result :Response. Look to the examples below for each number.

Negative Scenario: If we put number "5", or any number,It will be an error.

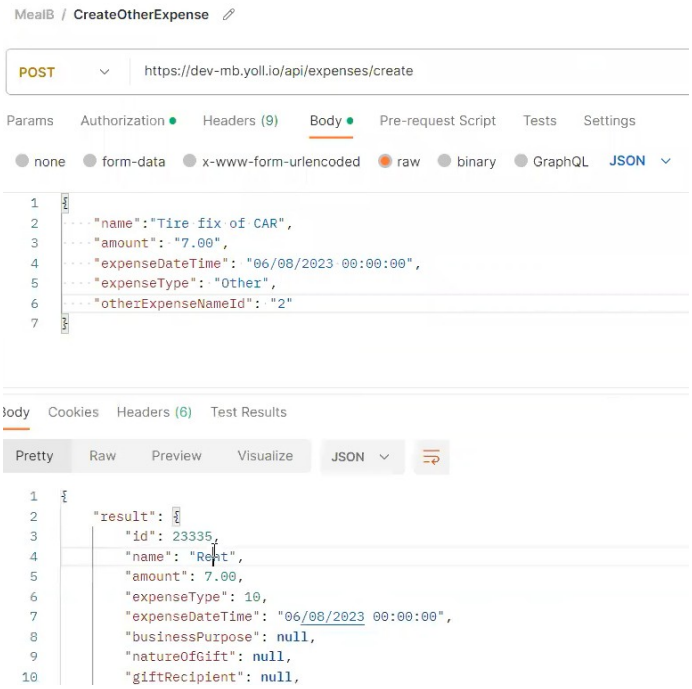


Result:

- **OtherExpenseNameID: "1",It will create : Name:"Electricity".**



- **OtherExpenseNameID: "2",It will create : Name:"Rent".**



- **OtherExpenseNameID: "2",It will create : Name:"Gas".**

MealB / CreateOtherExpense

POST https://dev-mb.yoll.io/api/expenses/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "name":"Tire fix of CAR",
    "amount": "7.00",
    "expenseDateTime": "06/08/2023 00:00:00",
    "expenseType": "Other",
    "otherExpenseNameId": "3"
}
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "result": {
        "id": 23336,
        "name": "Gas",
        "amount": 7.00,
        "expenseType": 10,
        "expenseDateTime": "06/08/2023 00:00:00",
        "businessPurpose": null,
        "natureOfGift": null,
        "giftRecipient": null,
        "vendorName": null,
```

- **OtherExpenseNameID: "4",It will create : Name:"Tire fix of CAR".**

MealB / CreateOtherExpense

POST https://dev-mb.yoll.io/api/expenses/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "name":"Tire fix of CAR",
    "amount": "7.00",
    "expenseDateTime": "06/08/2023 00:00:00",
    "expenseType": "Other",
    "otherExpenseNameId": "4"
}
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "result": {
        "id": 23337,
        "name": "Tire fix of CAR",
        "amount": 7.00,
        "expenseType": 10,
        "expenseDateTime": "06/08/2023 00:00:00",
        "businessPurpose": null,
        "natureOfGift": null,
        "giftRecipient": null,
        "vendorName": null,
```

- **Negative Scenario :OtherExpenseNameID: "5",It will be an error :**

MealB / CreateOtherExpense

POST https://dev-mb.yoll.io/api/expenses/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body Cookies Headers (6) Test Results Status: 500 Internal Server Error

Pretty Raw Preview Visualize JSON


```
{
    "result": null,
    "targetUrl": null,
    "success": false,
    "error": {
        "code": 0,
        "message": "An internal error occurred during your request!",
        "details": null,
        "validationErrors": null
    },
    "unAuthorizedRequest": false,
```

Create Other Expense (ExpenseType ==Other)

API: MealB - Create Expense API
First try the scenarios in Postman manually, then automate them

Requirements:
1. Create Request data from Java Object
2. Convert it to String and pass that string variable into body() method
3. Use Assertions to validate your expected result against actual result

Story: When expense type == Other then otherExpenseNameId is required element
(Note: otherExpenseNameId takes value between 1-4)

Sample request body:

```
{
    "name": "Disneyland ticket",
    "amount": 45475.00,
    "expenseDateTime": "01/28/2021 00:00:00",
    "expenseType": "Other",
    "otherExpenseNameId": "1"
}
```

Scenario 1: otherExpenseNameId :1
1.Submit a request where expense type == Other and otherExpenseNameId ==1
2. validate status code is 200
3. Validate if value of "name" in response is "Electricity"

Scenario 2: otherExpenseNameId :2
1.Submit a request where expense type == Other and otherExpenseNameId ==2
2. validate status code is 200
3. Validate if value of "name" in response is "Rent"

Scenario 3: otherExpenseNameId :3
1.Submit a request where expense type == Other and otherExpenseNameId ==3
2. validate status code is 200
3. Validate if value of "name" in response is "Gas"

Scenario 4: otherExpenseNameId :4

1.Submit a request where expense type == Other and otherExpenseNameId ==4

2. validate status code is 200

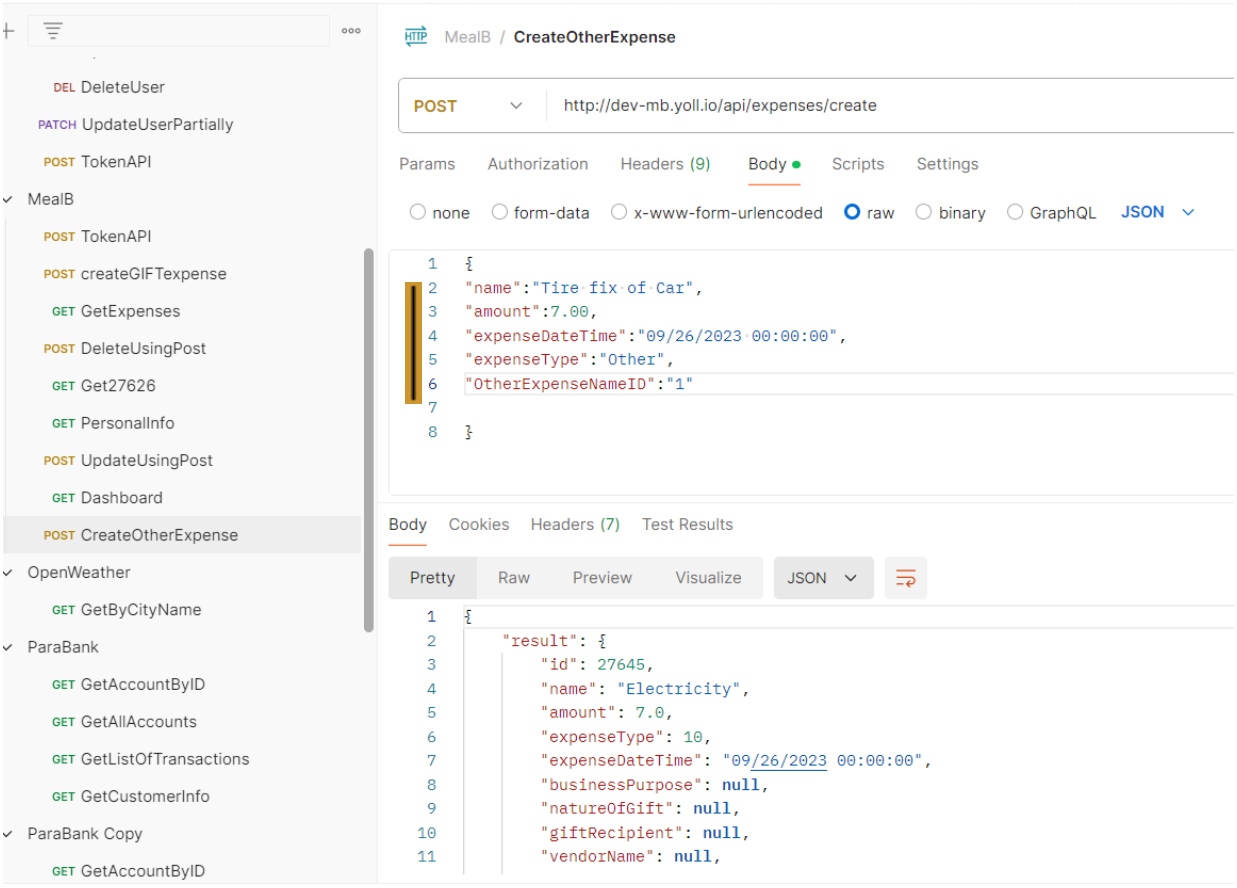
3. Validate if value of "name" in response matches with value of element "name" in request body

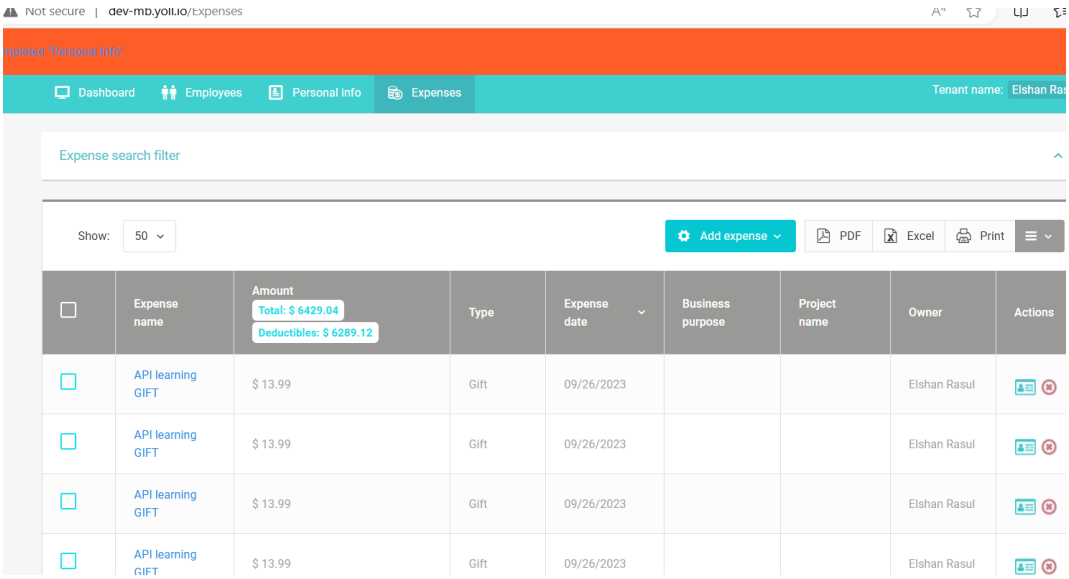
Scenario 5: otherExpenseNameId :0, 5,6,7 ...

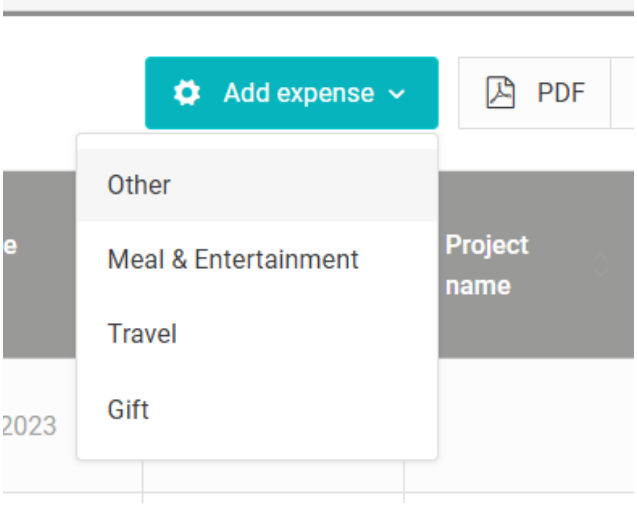
1.Submit a request where expense type == Other and otherExpenseNameId has different value

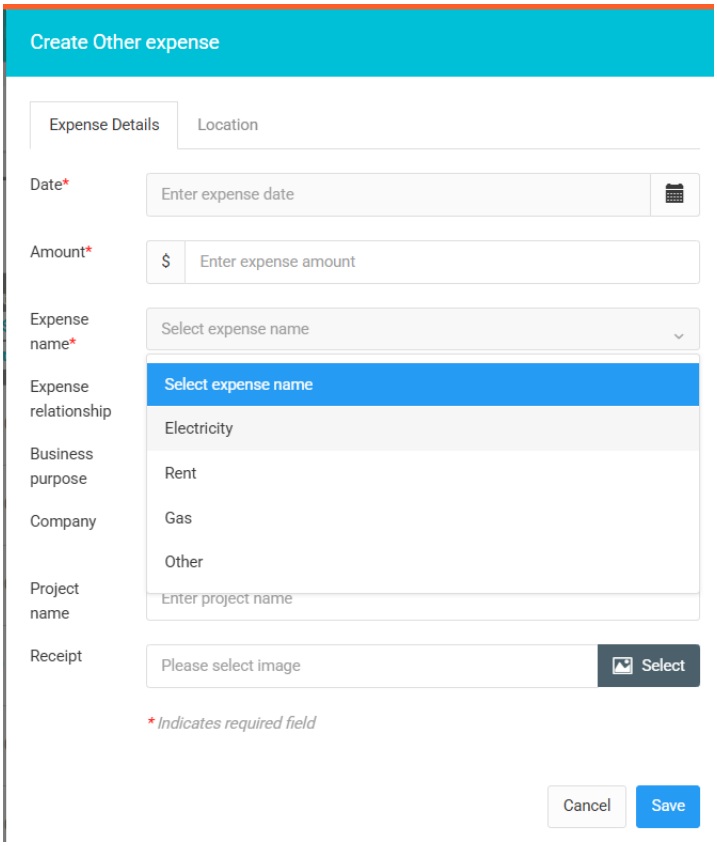
2. validate status code is 500

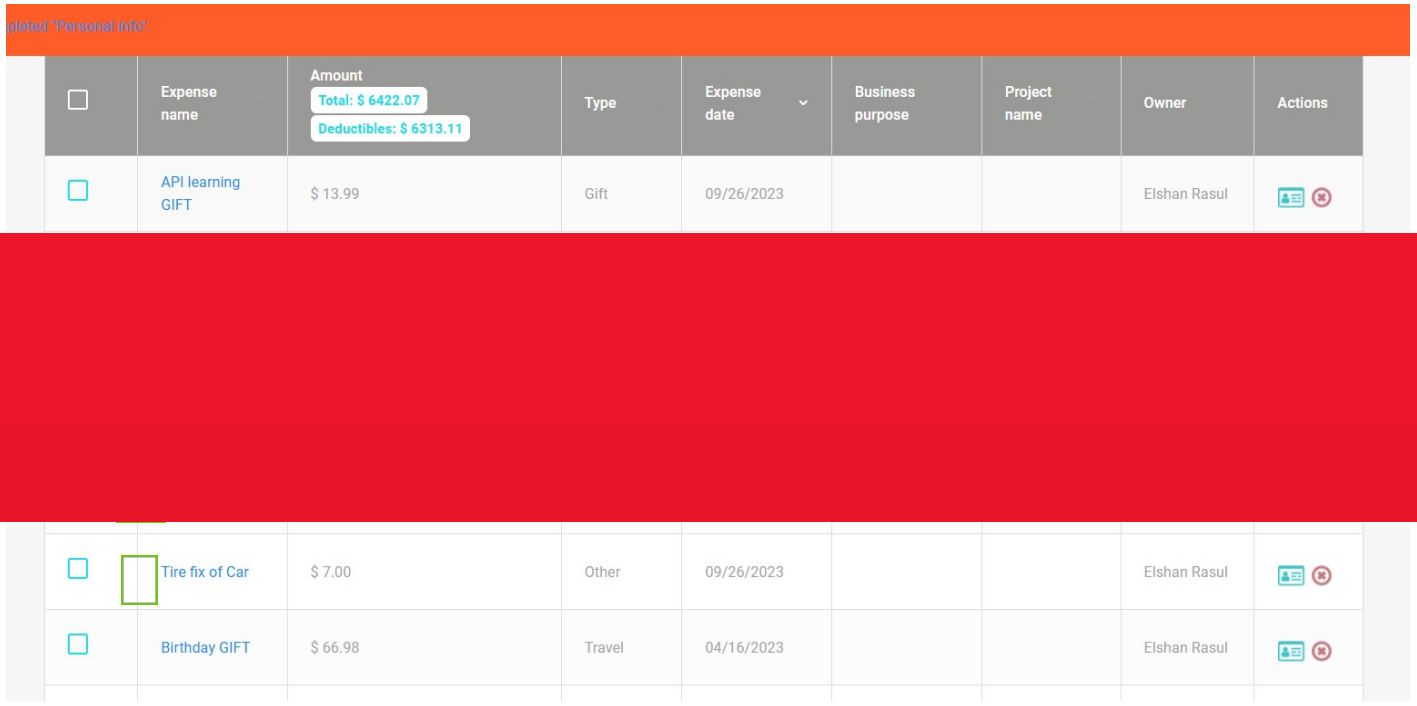
3.Validate error message: An internal error occurred during your request!




Not secure | dev-mb.yoll.io/Expenses
Dashboard
Employees
Personal info
Expenses
Tenant name:
Expense search filter
Show: 50
Add expense
PDF
Excel
Print
Expense name
Amount
Total: $ 6429.04
Deductibles: $ 6289.12
Type
Expense date
Business purpose
Project name
Owner
Actions
API learning GIFT
$ 13.99
Gift
09/26/2023
Elshan Rasul


Add expense
PDF
Other
Meal & Entertainment
Travel
Gift
Project name
2023





1- Create **OtherExpensePayloadBuilder** class under the `package: payloadBuilder;`

2-Create **Getters** and **Setters**

3-Create **CreateOtherExpense** Class under the **package** : **homework.**

4-Create FunctionLibray Class under the package : utils.
The class contains `getToken()`, `createTokenPayload()`, `convertObjToString()` Methods.

5-Extends CreateOtherExpense **from** FunctionLibrary.
Use **convertObjToString** method to convert payload objects to String format.

6- Use `@Ignore` `@Test` **to run only one Test, and ignore the other ones.**

relaxedHTTPSValidation is used with an URL that starts with :https.

If we starts an URL with http, we do not need to use it.

We will use two methods:

First method:

- CreateOtherExpense Class extended from FunctionLibrary. It has a repeated Coding.
- OtherExensePayloadBuilder Class for Getters and setters.
- FunctionLibrary Class containing convertObjToString method, and **`getToken()`, `createTokenPayload()` for creating the token.**

```
package payloadBuilder;

public class OtherExpensePayloadBuilder {
	private String name;
	private String amount;
	private String expenseDateTime;
	private String expenseType;
	private String OtherExpenseNameID;

	public String getAmount() {
		return amount;
	}
	public void setAmount(String amount) {
		this.amount = amount;
	}
	public String getExpenseDateTime() {
		return expenseDateTime;
	}
```

```
	public void setExpenseDateTime(String expenseDateTime) {
		this.expenseDateTime = expenseDateTime;
	}
	public String getExpenseType() {
		return expenseType;
	}
	public void setExpenseType(String expenseType) {
		this.expenseType = expenseType;
	}
	public String getOtherExpenseNameID() {
		return OtherExpenseNameID;
	}
	public void setOtherExpenseNameID(String otherExpenseNameID) {
		OtherExpenseNameID = otherExpenseNameID;
	}

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

}

package utils;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.TokenPayloadBuilder;

public class FunctionLibrary {
	String tokenUrl = "http://dev-mb.yoll.io/api/tokenauth/authenticate";

	/**
	 * @author ElshanRasulov This method submits POST request to create token and
	 *         retrieve token from response and return
	 */
	public String getToken() {
		Response response =
given().body(createTokenPayload()).contentType(ContentType.JSON).when().post(tokenUrl);
		response.prettyPrint();
		assertEquals(200, response.getStatusCode());
//extract "accessToken" from response
		String token = JsonPath.read(response.asString(), "$.result.accessToken");
		return token;
	}

	/**
	 * @author ElshanRasulov This method create token request payload data and
	 *         convert it to string and return
	 */
	public String createTokenPayload() {
		TokenPayloadBuilder payload = new TokenPayloadBuilder();
		payload.setUsernameOrEmailAddress("ElshanRasul");
```

```
		payload.setPassword("Elshan123");
		return convertObjToString(payload);
	}

	/**
	 * @author ElshanRasulov This method convert payload objects to String format
	 */
	public String convertObjToString(Object obj) {
		String strPayload = null;
		try {
			ObjectMapper objMap = new ObjectMapper();
			strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(obj);
			System.out.println(strPayload);
		} catch (JsonProcessingException e) {
			e.printStackTrace();
		}
		return strPayload;
	}
}
```

One Class containing the five Scenarios :

The Coding is repeated on each Scenario

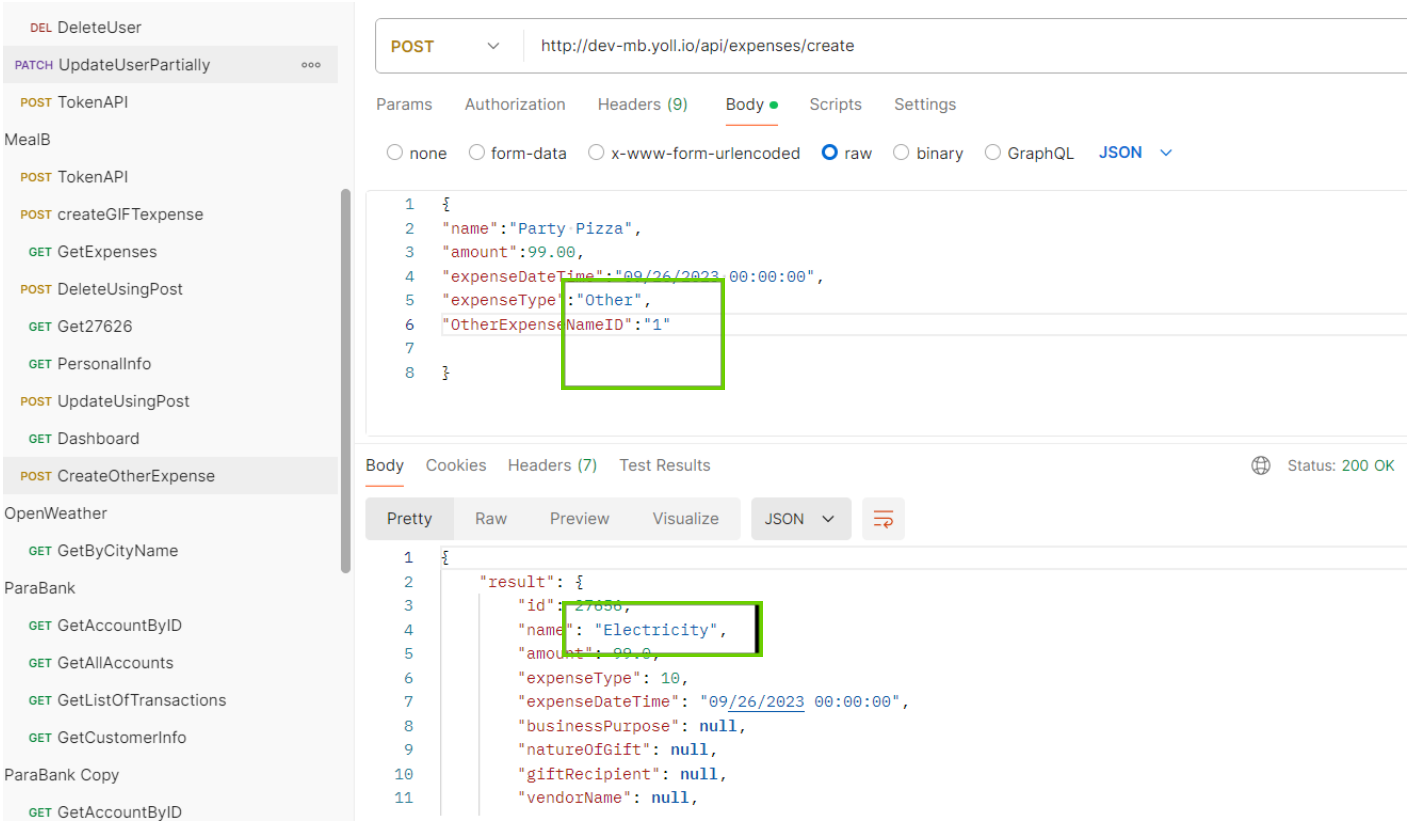
```
package homework;

import static org.junit.Assert.assertEquals;

import org.junit.Ignore;
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.OtherExpensePayloadBuilder;
import utils.FunctionLibrary;

public class CreateOtherExpense extends FunctionLibrary {
	@Ignore
	@Test
	public void scenario1() {
//create request payload
		OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
		payload.setName("Party Pizza");
		payload.setAmount("22.95");
		payload.setExpenseDateTime("09/26/2023 04:30:12");
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID("1");
		String strPayload = convertObjToString(payload);
// submit POST request
		String url = "http://dev-mb.yoll.io/api/expenses/create";
		Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
				.contentType(ContentType.JSON).when().post(url);
		response.prettyPrint();
// validate status code is 200
		assertEquals(200, response.getStatusCode());
//Validate if value of "name" in response is "Electricity"
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
```

```
            assertEquals("Electricity", name);
      }

      @Ignore
      @Test
      public void scenario2() {
            // create request payload
            OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
            payload.setName("Party Pizza");
            payload.setAmount("22.95");
            payload.setExpenseDateTime("09/26/2023 04:30:12");
            payload.setExpenseType("Other");
            payload.setOtherExpenseNameID("2");
            String strPayload = convertObjToString(payload);
            // submit POST request
            String url = "http://dev-mb.yoll.io/api/expenses/create";
            Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
                        .contentType(ContentType.JSON).when().post(url);
            response.prettyPrint();
            // validate status code is 200
            assertEquals(200, response.getStatusCode());
            // Validate if value of "name" in response is "Rent"
            String name = JsonPath.read(response.asString(), "$.result.name");
            System.out.println("Expense name:" + name);
            assertEquals("Rent", name);
      }

      @Ignore
      @Test
      public void scenario3() {
            // create request payload
            OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
            payload.setName("Party Pizza");
            payload.setAmount("22.95");
            payload.setExpenseDateTime("09/26/2023 04:30:12");
            payload.setExpenseType("Other");
            payload.setOtherExpenseNameID("3");
            String strPayload = convertObjToString(payload);
            // submit POST request
            String url = "http://dev-mb.yoll.io/api/expenses/create";
            Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
                        .contentType(ContentType.JSON).when().post(url);
            response.prettyPrint();
            // validate status code is 200
            assertEquals(200, response.getStatusCode());
            // Validate if value of "name" in response is "Gas"
            String name = JsonPath.read(response.asString(), "$.result.name");
            System.out.println("Expense name:" + name);
            assertEquals("Gas", name);
      }

      @Ignore
      @Test
      public void scenario4() {
            // create request payload
            OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
            payload.setName("Party Pizza");
            payload.setAmount("22.95");
```

```
            payload.setExpenseDateTime("09/26/2023 04:30:12");
            payload.setExpenseType("Other");
            payload.setOtherExpenseNameID("4");
            String strPayload = convertObjToString(payload);
            // submit POST request
            String url = "http://dev-mb.yoll.io/api/expenses/create";
            Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
                        .contentType(ContentType.JSON).when().post(url);
            response.prettyPrint();
            // validate status code is 200
            assertEquals(200, response.getStatusCode());
            // Validate if value of "name" in response is value of name
            String name = JsonPath.read(response.asString(), "$.result.name");
            System.out.println("Expense name:" + name);
            assertEquals("Party Pizza", name);
            assertEquals(payload.getName(), name);
      }

      @Test
      public void scenario5() {
            // create request payload
            OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
            payload.setName("Party Pizza");
            payload.setAmount("22.95");
            payload.setExpenseDateTime("09/26/2023 04:30:12");
            payload.setExpenseType("Other");
            payload.setOtherExpenseNameID("5");
            String strPayload = convertObjToString(payload);
            // submit POST request
            String url = "http://dev-mb.yoll.io/api/expenses/create";
            Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
                        .contentType(ContentType.JSON).when().post(url);
            response.prettyPrint();
            // validate status code is 500
            assertEquals(500, response.getStatusCode());
            // Validate error message:An internal error occurred during your request!
            String error = JsonPath.read(response.asString(), "$.error.message");
            System.out.println(error);
            assertEquals("An internal error occurred during your request!", error);
      }

}
```

Scenarios one by one with result of each one:

Scenario 1:



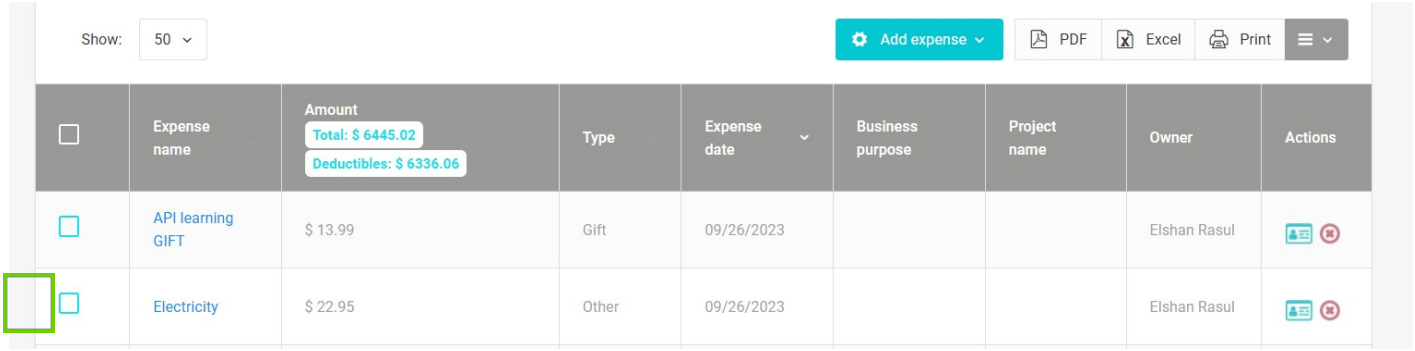

```java
package homework;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.OtherExpensePayloadBuilder;
import utils.FunctionLibrary;
public class CreateOtherExpense extends FunctionLibrary{
@Test
public void scenario1() {
//create request payload
OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
payload.setName("Party Pizza");
payload.setAmount("22.95");
payload.setExpenseDateTime("09/26/2023 04:30:12");
payload.setExpenseType("Other");
payload.setOtherExpenseNameID("1");
String strPayload=convertObjToString(payload);
// submit POST request
String url="http://dev-mb.yoll.io/api/expenses/create";
Response response=given()
.header("Authorization", "Bearer "+getToken())
.body(strPayload)
.contentType(ContentType.JSON)
.when()
.post(url);
response.prettyPrint();
// validate status code is 200
assertEquals(200, response.getStatusCode());
//Validate if value of "name" in response is "Electricity"
String name=JsonPath.read(response.asString(), "$.result.name");
```

```
System.out.println("Expense name:"+name);
assertEquals("Electricity",name );
}
}
```

Result Scenario 1:

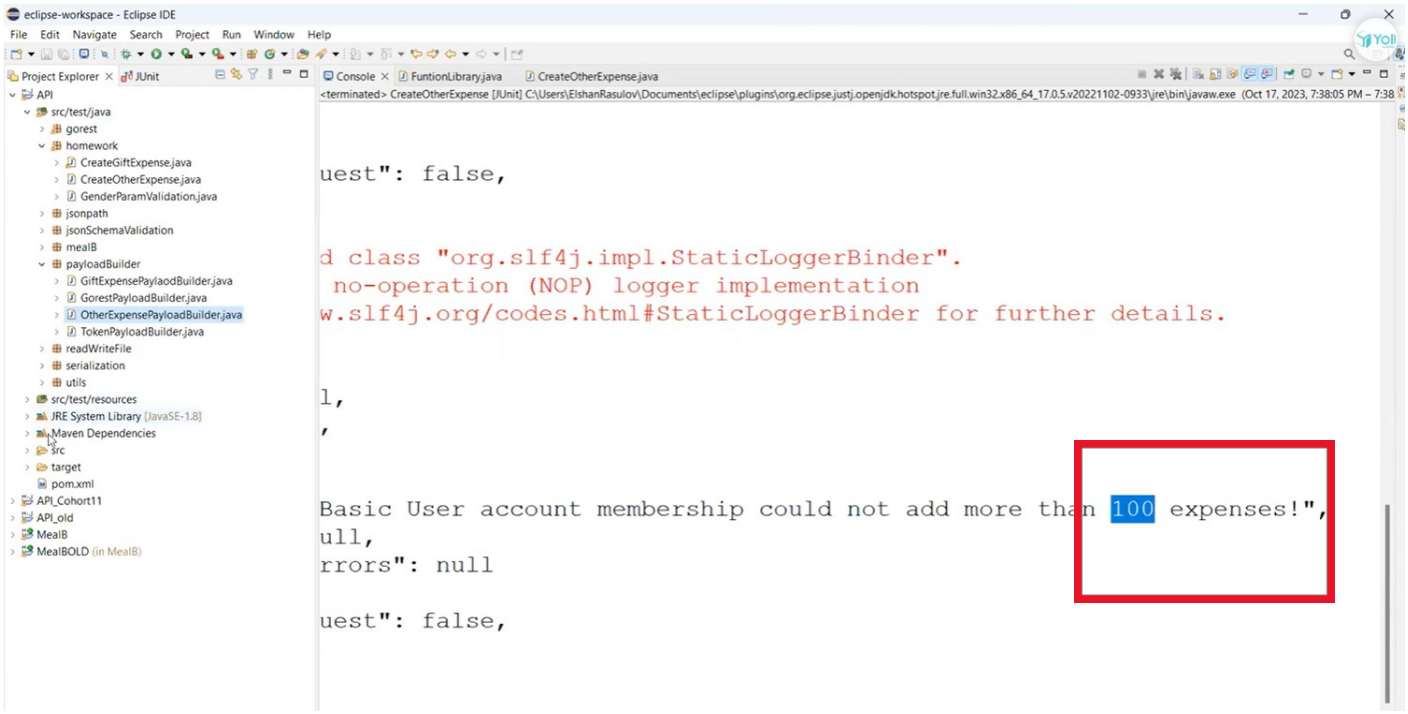
```
{
  "name" : "Party Pizza",
  "amount" : "22.95",
  "expenseDateTime" : "09/26/2023 04:30:12",
  "expenseType" : "Other",
  "otherExpenseNameID" : "1"
}
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "
.
.
.
.
",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
        "id": 27650,
        "name": "Electricity",
        "amount": 22.95,
        "expenseType": 10,

    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
Expense name:Electricity
```

We can see the result on Java and MealB



We are expecting 200 to validate status code.

If we got 500 and this message:"Basic User account membership could not add more than 100 expenses!", then we need to delete some recordes because 100 is the Maximum.



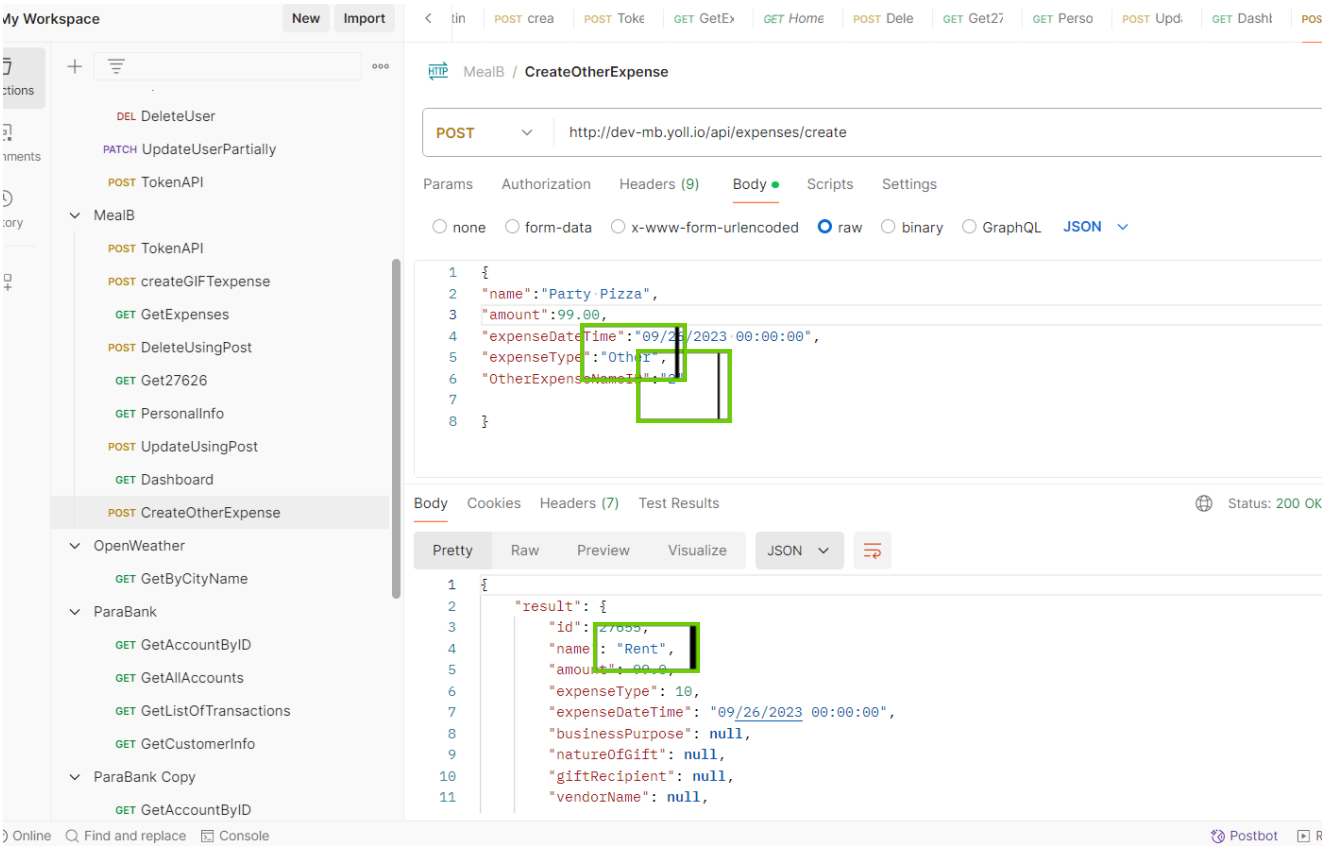
SCENARIOS 2, 3, 4, 5:

1 = Electricity 2 = Rent 3=Gas

4 =name; means that we need to get the name written on name, and use: getName().

5= if using number 5 and above, we have to get an error message: "An internal error occurred during your request!" by using : "$.error.message");

Scenario 2:

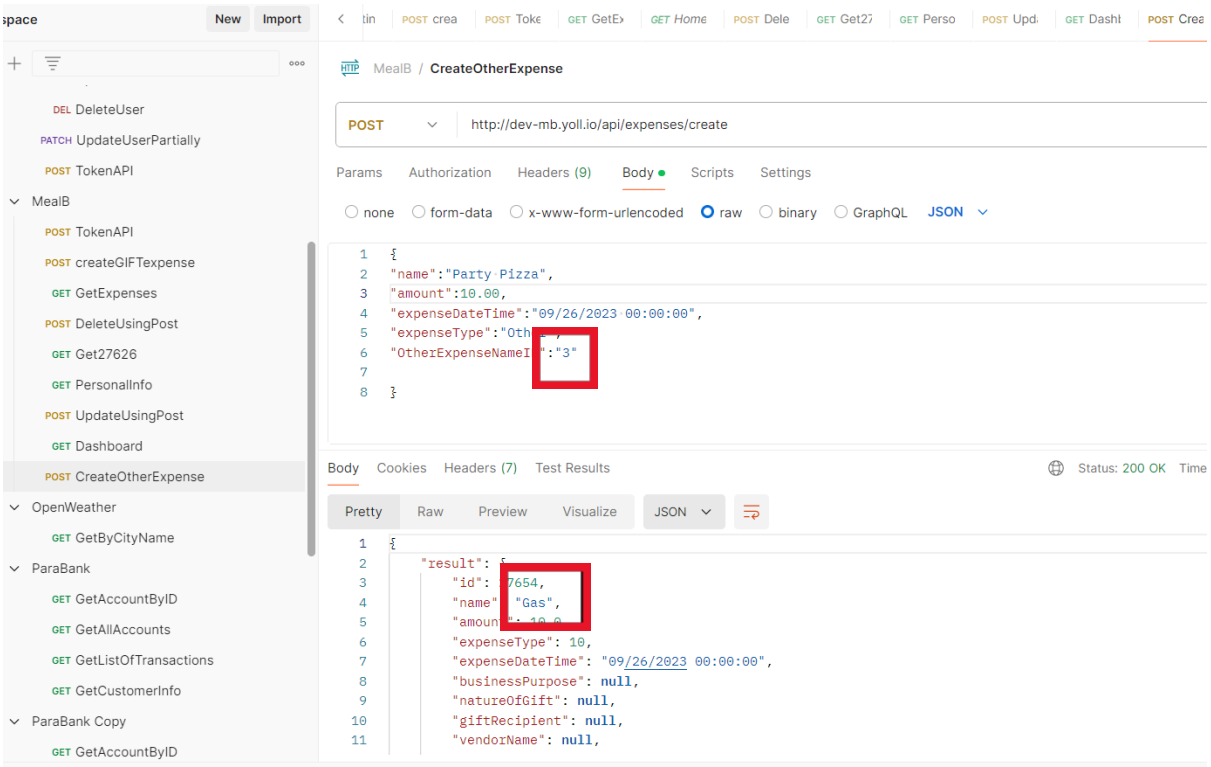


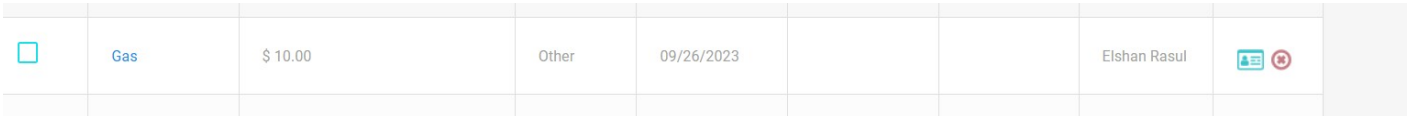

```
@Test
	public void scenario2() {
		// create request payload
		OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
		payload.setName("Party Pizza");
		payload.setAmount("22.95");
		payload.setExpenseDateTime("09/26/2023 04:30:12");
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID("2");
		String strPayload = convertObjToString(payload);
		// submit POST request
		String url = "http://dev-mb.yoll.io/api/expenses/create";
		Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
				.contentType(ContentType.JSON).when().post(url);
		response.prettyPrint();
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
		// Validate if value of "name" in response is "Rent"
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		assertEquals("Rent", name);
	}
```

Result Scenario 2:


```
{
  "name" : "Party Pizza",
  "amount" : "22.95",
  "expenseDateTime" : "09/26/2023 04:30:12",
  "expenseType" : "Other",
  "otherExpenseNameID" : "2"
}
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "eyJhb
.
.
.
{
    "result": {
        "id": 27653,
        "name": "Rent",
        "amount": 22.95,
.
.
.
  "__abp": true
}
Expense name:Rent
```


Scenario 3:





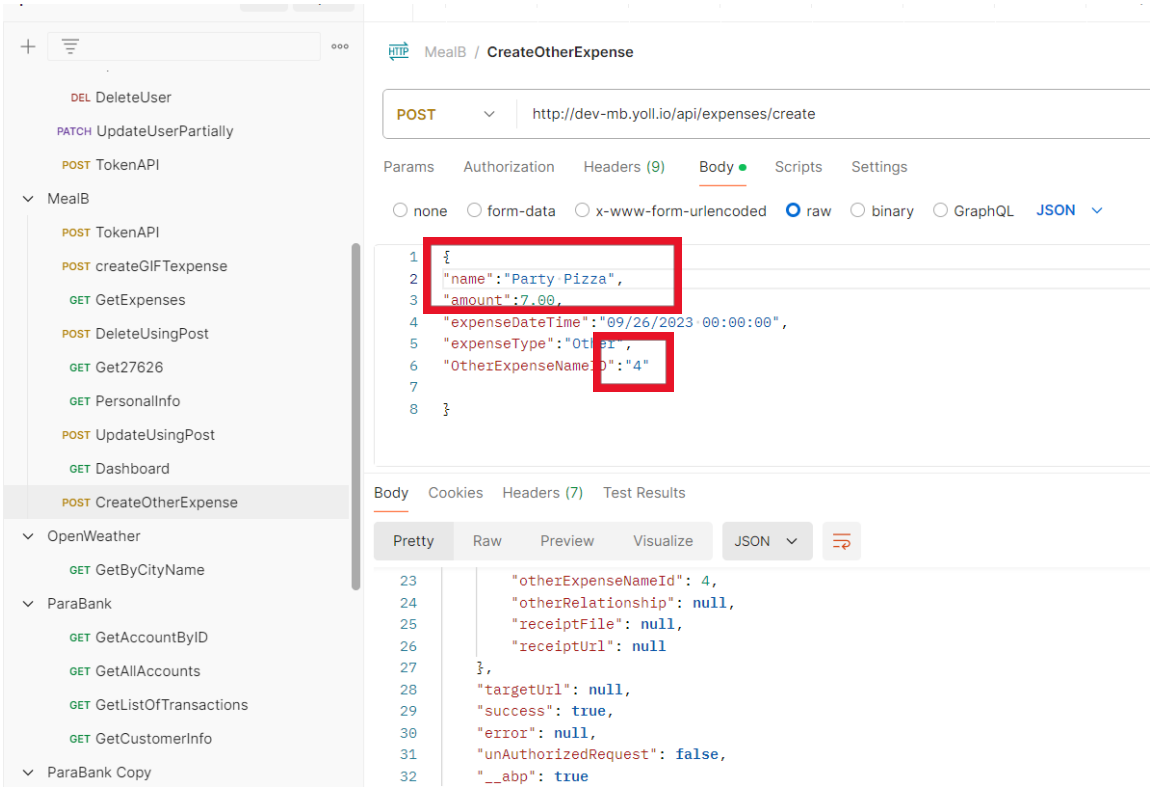

```
	@Test
	public void scenario3() {
		// create request payload
		OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
		payload.setName("Party Pizza");
		payload.setAmount("22.95");
		payload.setExpenseDateTime("09/26/2023 04:30:12");
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID("3");
		String strPayload = convertObjToString(payload);
		// submit POST request
		String url = "http://dev-mb.yoll.io/api/expenses/create";
		Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
				.contentType(ContentType.JSON).when().post(url);
		response.prettyPrint();
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
		// Validate if value of "name" in response is "Gas"
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		assertEquals("Gas", name);
	}
```

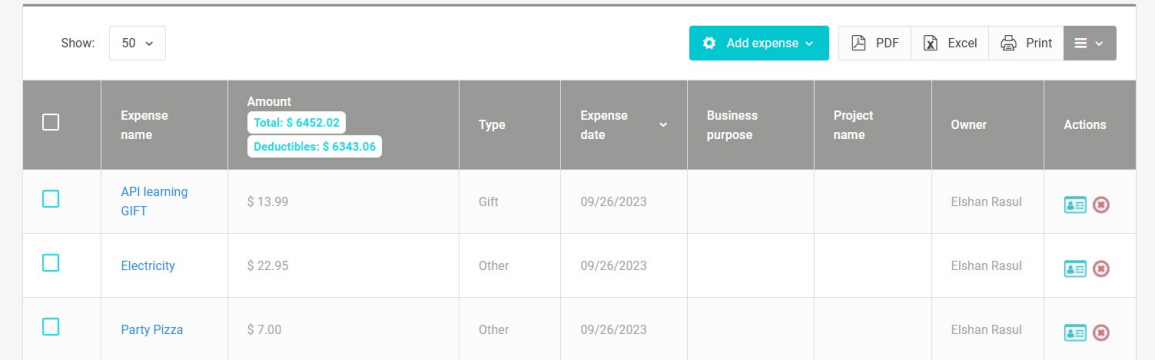
Result Scenario 3:

```
{
  "name" : "Party Pizza",
  "amount" : "22.95",
  "expenseDateTime" : "09/26/2023 04:30:12",
  "expenseType" : "Other",
```

```
  "otherExpenseNameID" : "3"
}
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "eyJhb
.
.
.
{
    "result": {
        "id": 27653,
        "name": "Gas",
        "amount": 22.95,
.
.
.
  "__abp": true
}
Expense name:Gas
```

Scenario 4:





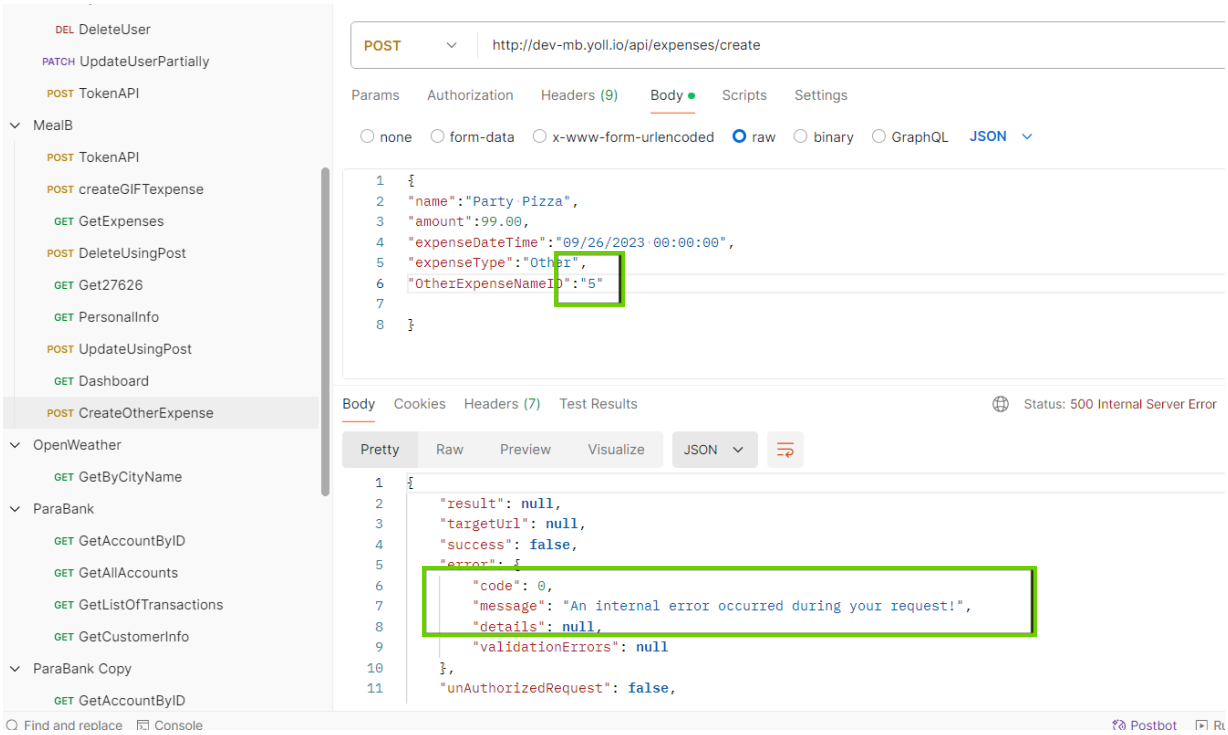

```
	We need to use getName()
	@Test
	public void scenario4() {
		// create request payload
		OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
		payload.setName("Party Pizza");
		payload.setAmount("22.95");
		payload.setExpenseDateTime("09/26/2023 04:30:12");
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID("4");
		String strPayload = convertObjToString(payload);
		// submit POST request
		String url = "http://dev-mb.yoll.io/api/expenses/create";
		Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
				.contentType(ContentType.JSON).when().post(url);
		response.prettyPrint();
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
		// Validate if value of "name" in response is value of name
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		// We can use this:
		assertEquals("Party Pizza", name);
		// Or:
		assertEquals(payload.getName(), name);
	}
```

Scenario 4 Result:

```
{
  "name" : "Party Pizza",
  "amount" : "22.95",
  "expenseDateTime" : "09/26/2023 04:30:12",
  "expenseType" : "Other",
  "otherExpenseNameID" : "4"
}
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
.
.
```

```
.
.
  "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
Expense name:Party Pizza
```

Scenario 5




```
@Test
	public void scenario5() {
		// create request payload
		OtherExpensePayloadBuilder payload = new OtherExpensePayloadBuilder();
		payload.setName("Party Pizza");
		payload.setAmount("22.95");
		payload.setExpenseDateTime("09/26/2023 04:30:12");
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID("5");
		String strPayload = convertObjToString(payload);
		// submit POST request
		String url = "http://dev-mb.yoll.io/api/expenses/create";
		Response response = given().header("Authorization", "Bearer " +
getToken()).body(strPayload)
				.contentType(ContentType.JSON).when().post(url);
		response.prettyPrint();
		// validate status code is 500
		assertEquals(500, response.getStatusCode());
		// Validate error message:An internal error occurred during your request!
		String error = JsonPath.read(response.asString(), "$.error.message");
		System.out.println(error);
		assertEquals("An internal error occurred during your request!", error);
	}
```

Second method:by avoiding the repeated coding on the previous method

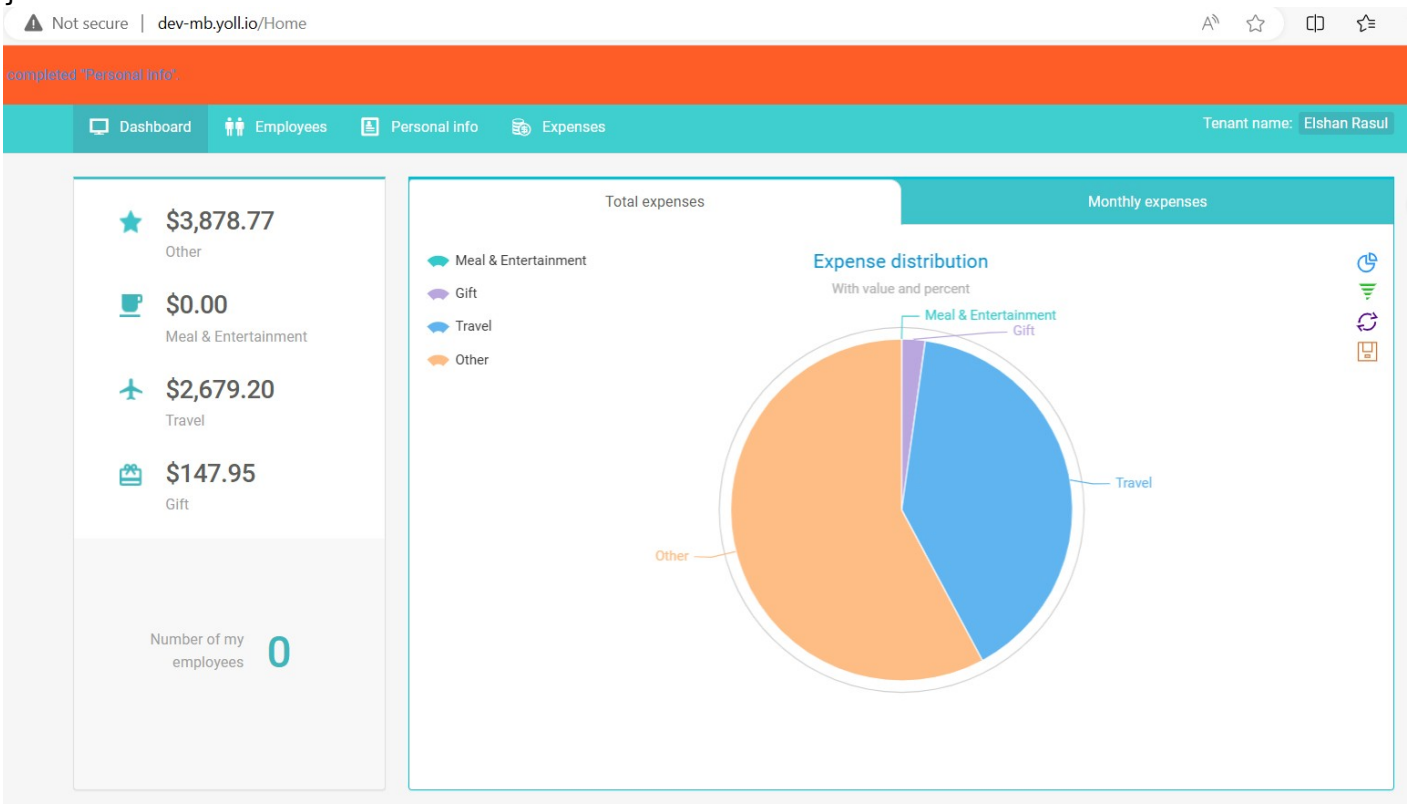
Create CreateOtherExpenseWithMethod Class extended from FunctionLibrary using: - createOtherExpensePayload() to create request payload. - createExpense() to submit the Post request.

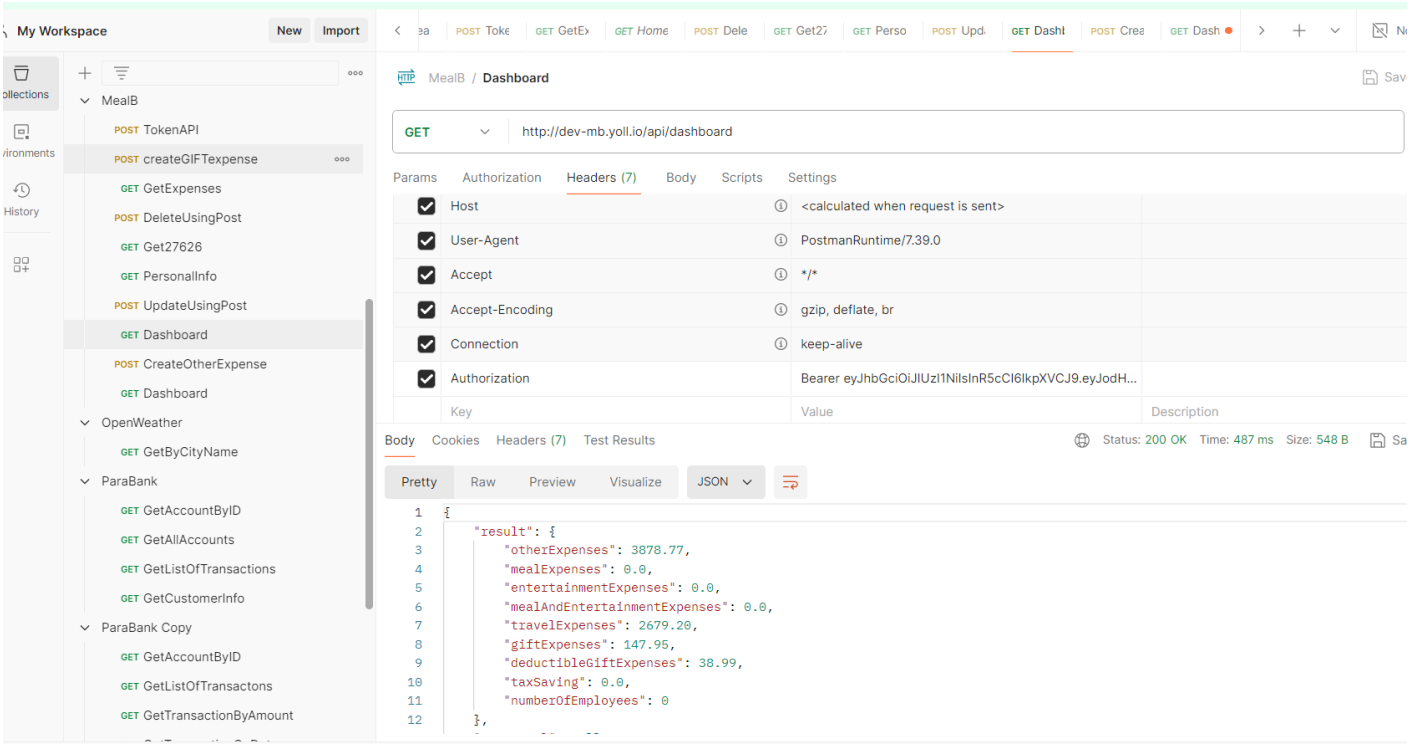
```
package homework;

import static org.junit.Assert.assertEquals;
import org.junit.Ignore;
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.OtherExpensePayloadBuilder;
import utils.FunctionLibrary;

public class CreateOtherExpenseWithMethod extends FunctionLibrary {
	@Test
	public void scenario5() {
		// create request payload
		// OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
		// payload.setName("Party Pizza");
		// payload.setAmount("22.95");
		// payload.setExpenseDateTime("09/26/2023 04:30:12");
		// payload.setExpenseType("Other");
		// payload.setOtherExpenseNameId("5");
		//
		// String strPayload=convertObjToString(payload);
		String strPayload = createOtherExpensePayload("Party Pizza", "22.95",
"09/26/2023 04:30:12", "5");
		// submit POST request
		// String url="http://dev-mb.yoll.io/api/expenses/create";
		// Response response=given()
		// .header("Authorization", "Bearer "+getToken())
		// .body(strPayload)
		// .contentType(ContentType.JSON)
		// .when()
		// .post(url);
		// response.prettyPrint();
		Response response = createExpense(strPayload);
		// validate status code is 500
		assertEquals(500, response.getStatusCode());
		// Validate error message:An internal error occurred during your request!
		String error = JsonPath.read(response.asString(), "$.error.message");
		System.out.println(error);
```

```
		assertEquals("An internal error occurred during your request!", error);
	}

	@Ignore
	@Test
	public void scenario4() {
		// create request payload
		// OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
		// payload.setName("Party Pizza");
		// payload.setAmount("22.95");
		// payload.setExpenseDateTime("09/26/2023 04:30:12");
		// payload.setExpenseType("Other");
		// payload.setOtherExpenseNameId("4");
		//
		// String strPayload=convertObjToString(payload);
		String strPayload = createOtherExpensePayload("Party Pizza", "22.95",
"09/26/2023 04:30:12", "4");
		// submit POST request
		// String url="http://dev-mb.yoll.io/api/expenses/create";
		// Response response=given()
		// .header("Authorization", "Bearer "+getToken())
		// .body(strPayload)
		// .contentType(ContentType.JSON)
		// .when()
		// .post(url);
		// response.prettyPrint();
		Response response = createExpense(strPayload);
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
		// Validate if value of "name" in response is value of name
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		assertEquals("Party Pizza", name);
	}

	@Ignore
	@Test
	public void scenario3() {
		// create request payload
		// OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
		// payload.setName("Party Pizza");
		// payload.setAmount("22.95");
		// payload.setExpenseDateTime("09/26/2023 04:30:12");
		// payload.setExpenseType("Other");
		// payload.setOtherExpenseNameId("3");
		//
		// String strPayload=convertObjToString(payload);
		String strPayload = createOtherExpensePayload("Party Pizza", "22.95",
"09/26/2023 04:30:12", "3");
		// submit POST request
		// String url="http://dev-mb.yoll.io/api/expenses/create";
		// Response response=given()
		// .header("Authorization", "Bearer "+getToken())
		// .body(strPayload)
		// .contentType(ContentType.JSON)
		// .when()
		// .post(url);
		// response.prettyPrint();
		Response response = createExpense(strPayload);
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
```

```
		// Validate if value of "name" in response is "Gas"
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		assertEquals("Gas", name);
	}

	@Ignore
	@Test
	public void scenario2() {
		// create request payload
		// OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
		// payload.setName("Party Pizza");
		// payload.setAmount("22.95");
		// payload.setExpenseDateTime("09/26/2023 04:30:12");
		// payload.setExpenseType("Other");
		// payload.setOtherExpenseNameId("2");
		//
		// String strPayload=convertObjToString(payload);
		String strPayload = createOtherExpensePayload("Party Pizza", "22.95",
"09/26/2023 04:30:12", "2");
		// submit POST request
		// String url="http://dev-mb.yoll.io/api/expenses/create";
		// Response response=given()
		// .header("Authorization", "Bearer "+getToken())
		// .body(strPayload)
		// .contentType(ContentType.JSON)
		// .when()
		// .post(url);
		// response.prettyPrint();
		Response response = createExpense(strPayload);
		// validate status code is 200
		assertEquals(200, response.getStatusCode());
		// Validate if value of "name" in response is "Rent"
		String name = JsonPath.read(response.asString(), "$.result.name");
		System.out.println("Expense name:" + name);
		assertEquals("Rent", name);
	}

	@Ignore
	@Test
	public void scenario1() {
		// create request payload
		// OtherExpensePayloadBuilder payload=new OtherExpensePayloadBuilder();
		// payload.setName("Party Pizza");
		// payload.setAmount("22.95");
		// payload.setExpenseDateTime("09/26/2023 04:30:12");
		// payload.setExpenseType("Other");
		// payload.setOtherExpenseNameId("1");
		//
		// String strPayload=convertObjToString(payload);
		String strPayload = createOtherExpensePayload("Party Pizza", "22.95",
"09/26/2023 04:30:12", "1");
		// submit POST request
		// String url="http://dev-mb.yoll.io/api/expenses/create";
		// Response response=given()
		// .header("Authorization", "Bearer "+getToken())
		// .body(strPayload)
		// .contentType(ContentType.JSON)
		// .when()
		// .post(url);
		// response.prettyPrint();
```

```
        Response response = createExpense(strPayload);
        // validate status code is 200
        assertEquals(200, response.getStatusCode());
        // Validate if value of "name" in response is "Electricity"
        String name = JsonPath.read(response.asString(), "$.result.name");
        System.out.println("Expense name:" + name);
        assertEquals("Electricity", name);
    }
}
```





Example with an error :

On dashboardResponseSchema.json file, we have:

```
"required": [
        "Expenses",
        "mealExpenses",
        "entertainmentExpenses",
        "mealAndEntertainmentExpenses",
        "travelExpenses",
        "giftExpenses",
        "deductibleGiftExpenses",
        "taxSaving",
        "numberOfEmployees"
      ]
```

After running : DashboardAPI.java, we got the result on console:

```
"result": [
        "otherExpenses",
```

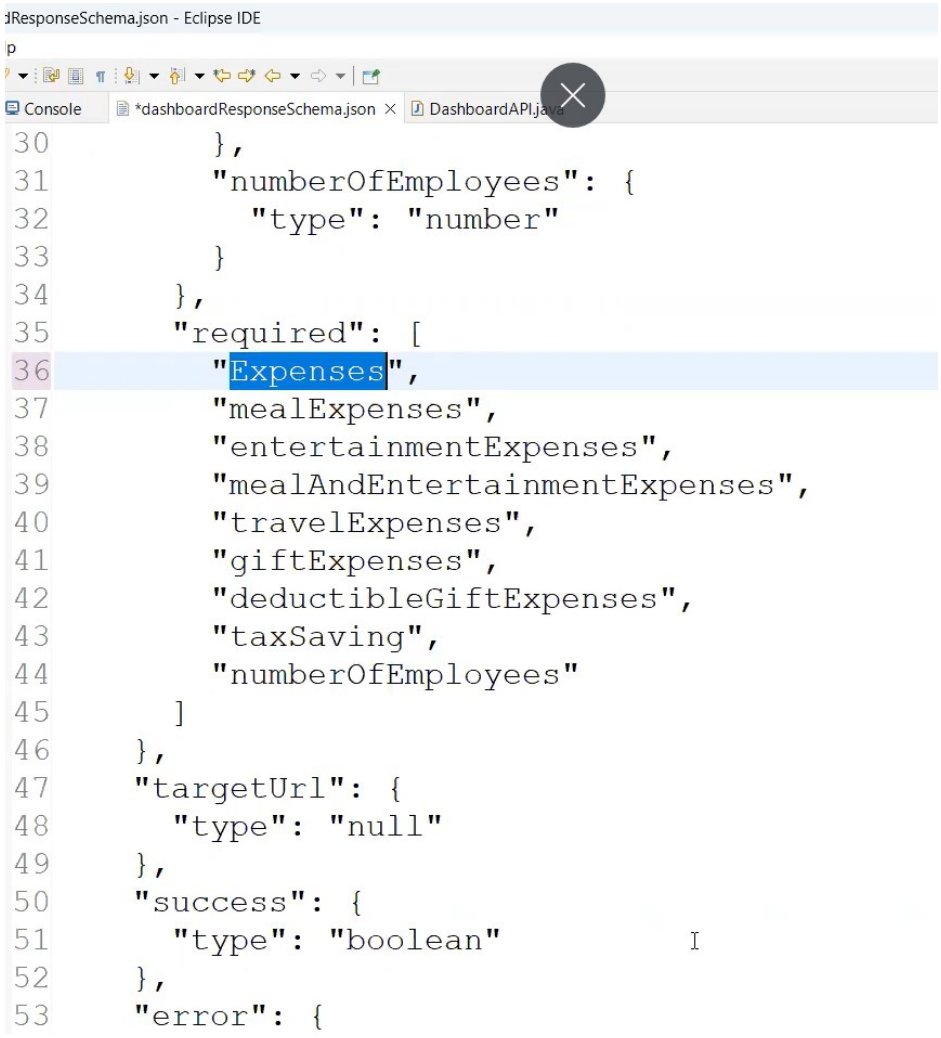
Instead of having the **expected required result** : **Expenses on** dashboardResponseSchema.json **file,** we have **the actual result on the reponse file** : **otherExpenses which results an error**.

- **Use this line for response validation on DashboardAPI.java:**

```
response.then().assertThat().body(matchesJsonSchemaInClasspath
("schema/dashboardResponseSchema.json"));
```

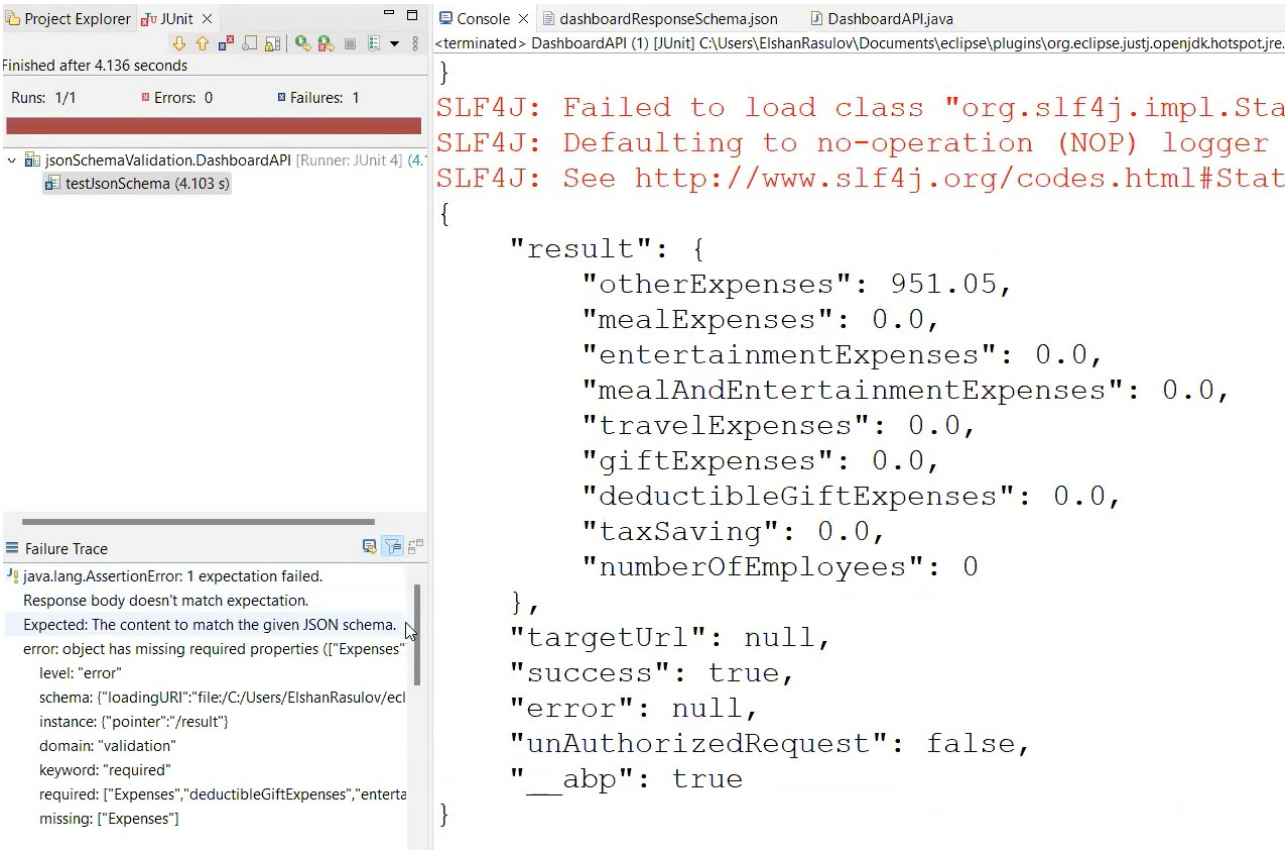
It verifies whether the response body follows the structure defined in the JSON schema file.

"schema/dashboardResponseSchema.json" is the schema file located in the resources/schema/ folder of the project.



On dashboardResponseSchema.json file, data are validated through this part:

```
"required": [
        "Expenses",
        "mealExpenses",
        "entertainmentExpenses",
        "mealAndEntertainmentExpenses",
        "travelExpenses",
        "giftExpenses",
        "deductibleGiftExpenses",
        "taxSaving",
        "numberOfEmployees"
      ]
```



Error: object has missing required properties(["Expenses"])

```
"result": [
        "otherExpenses",  // it supposed to be Expenses not otherExpenses
        "mealExpenses",
        "entertainmentExpenses",
        "mealAndEntertainmentExpenses",
        "travelExpenses",
        "giftExpenses",
        "deductibleGiftExpenses",
        "taxSaving",
        "numberOfEmployees"
      ]
```

- dashboardResponseSchema.json file (Request File) has the following data:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "object",
      "properties": {
        "otherExpenses": {
          "type": "number"
        },
        "mealExpenses": {
          "type": "number"
        },
```

```
        "entertainmentExpenses": {
          "type": "number"
        },
        "mealAndEntertainmentExpenses": {
          "type": "number"
        },
        "travelExpenses": {
          "type": "number"
        },
        "giftExpenses": {
          "type": "number"
        },
        "deductibleGiftExpenses": {
          "type": "number"
        },
        "taxSaving": {
          "type": "number"
        },
        "numberOfEmployees": {
          "type": "integer"
        }
      },
      "required": [
        "Expenses",
        "mealExpenses",
        "entertainmentExpenses",
        "mealAndEntertainmentExpenses",
        "travelExpenses",
        "giftExpenses",
        "deductibleGiftExpenses",
        "taxSaving",
        "numberOfEmployees"
      ]
    },
    "targetUrl": {
      "type": "null"
    },
    "success": {
      "type": "boolean"
    },
    "error": {
      "type": "null"
    },
    "unAuthorizedRequest": {
      "type": "boolean"
    },
    "__abp": {
      "type": "boolean"
    }
  },
  "required": [
    "result",
    "targetUrl",
    "success",
    "error",
    "unAuthorizedRequest",
    "__abp"
  ]
}
```

- DashboardAPI Java file extended from FunctionLibrary:

```
package jsonSchemaValidation;

import static io.restassured.RestAssured.given;
import static
io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspa
th;
import org.junit.Test;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class DashboardAPI extends FunctionLibrary {
     String url = "http://dev-mb.yoll.io/api/dashboard";

     @Test
     public void testJsonSchema() {
          String token = getToken();
// submit GET request for expenses
          Response repsonse = given().header("Authorization", "Bearer "
+ token).when().get(url);
          repsonse.prettyPrint();
//validate json schema


repsonse.then().assertThat().body(matchesJsonSchemaInClasspath("schema/
dashboardResponseSchema.json"));
     }
}
```

After running DashboardAPI.java, we got this response on the Console:

Result on Console:

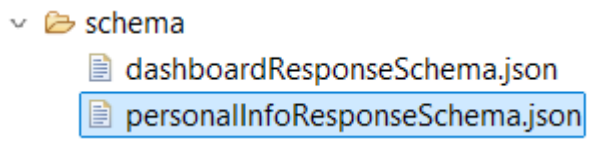

```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
```

```
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "99999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": null,
            "city": "Rabat",
            "postalCode": null,
            "tenantName": "",
            "accountType": 30,
            "birthDate": null,
            "businessType": null,
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": null,
            "countryStateId": null
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
        "otherExpenses": 3878.77,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 2679.20,
        "giftExpenses": 147.95,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```

Task:

validate the json file to see the result of phonenumber. It supposed to have a string type, but on json file it is a number type which will cause an error.

- **Create personalInfoResponseSchema.json under schema folder.**



- **Copy the data from the Get Method on Postman on the Generator** https://www.liquid-technologies.com/online-json-to-schema-converter**, and convert json to json scema, then paste it on: personalInfoResponseSchema.json file. Save it.**
  Json Schema result for phoneNumber :

```
    "phoneNumber": {
      "type": "string"
    },
```

But we have on personalInfoResponseSchema.json file :

```
"phoneNumber": {
      "type": "number"
    },
```

Which will cause an error !

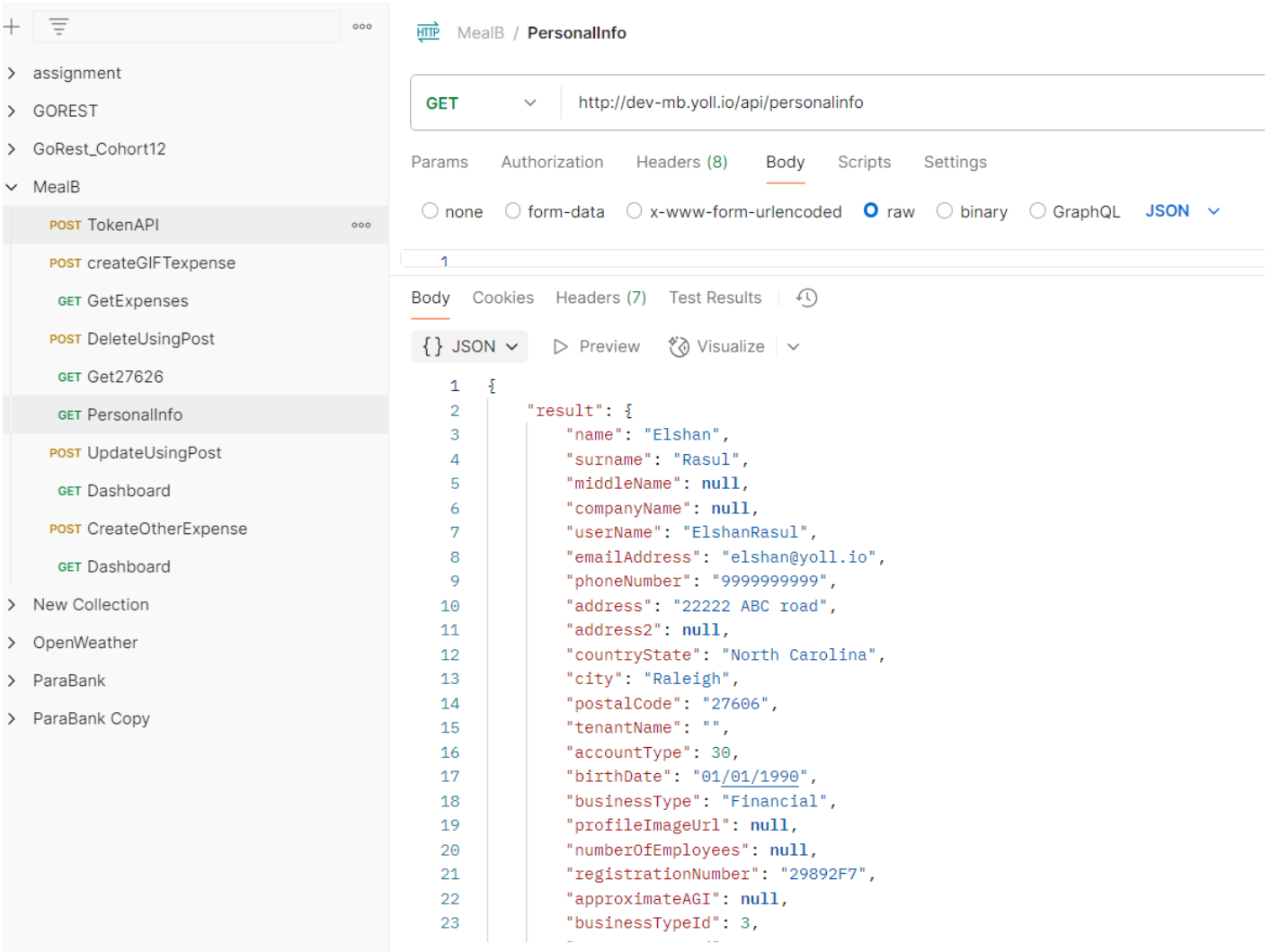
- **Create personalInfoAPI.java Class under jsonSchemaValidation package.**
- **PersonlaInfoAPI.java must be extended from FunctionLibrary.java.**
- **Run PersonalInfoAPI.java, and see the result on the console.**

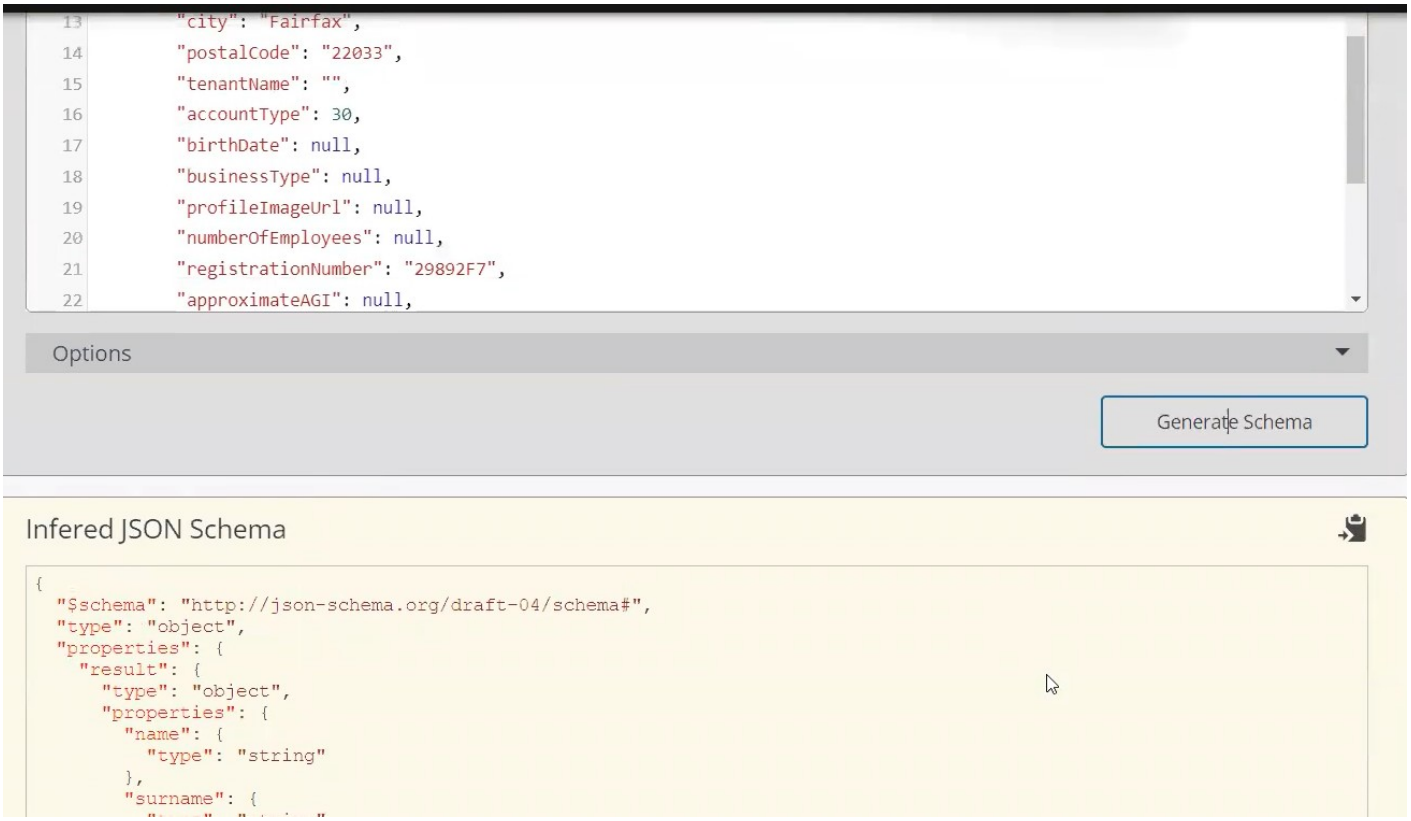
- **This line is used in API testing with RestAssured to validate if the response body matches a predefined JSON schema.**
- **Use this line for response validation:**

```
response.then().assertThat().body(matchesJsonSchemaInClasspath("schema/personalInfoResponseSchema.json"));
```

It verifies whether the response body follows the structure defined in the JSON schema file.

"schema/ personalInfoResponseSchema.json" is the schema file located in the resources/schema/ folder of the project.





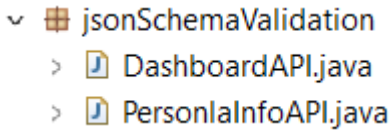
personalInfoResponseSchema.json file:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "surname": {
          "type": "string"
        },
        "middleName": {
          "type": "null"
        },
        "companyName": {
          "type": "null"
        },
        "userName": {
          "type": "string"
        },
        "emailAddress": {
          "type": "string"
        },
        "phoneNumber": {
          "type": "number"
        },
        "address": {
          "type": "string"
        },
```

```
    "address2": {
      "type": "null"
    },
    "countryState": {
      "type": "string"
    },
    "city": {
      "type": "string"
    },
    "postalCode": {
      "type": "string"
    },
    "tenantName": {
      "type": "string"
    },
    "accountType": {
      "type": "integer"
    },
    "birthDate": {
      "type": "string"
    },
    "businessType": {
      "type": "string"
    },
    "profileImageUrl": {
      "type": "null"
    },
    "numberOfEmployees": {
      "type": "null"
    },
    "registrationNumber": {
      "type": "string"
    },
    "approximateAGI": {
      "type": "null"
    },
    "businessTypeId": {
      "type": "integer"
    },
    "countryStateId": {
      "type": "integer"
    }
  },
  "required": [
    "name",
    "surname",
    "middleName",
    "companyName",
    "userName",
    "emailAddress",
    "phoneNumber",
    "address",
    "address2",
    "countryState",
    "city",
    "postalCode",
    "tenantName",
    "accountType",
    "birthDate",
    "businessType",
    "profileImageUrl",
```

```
        "numberOfEmployees",
        "registrationNumber",
        "approximateAGI",
        "businessTypeId",
        "countryStateId"
      ]
    },
    "targetUrl": {
      "type": "null"
    },
    "success": {
      "type": "boolean"
    },
    "error": {
      "type": "null"
    },
    "unAuthorizedRequest": {
      "type": "boolean"
    },
    "__abp": {
      "type": "boolean"
    }
  },
  "required": [
    "result",
    "targetUrl",
    "success",
    "error",
    "unAuthorizedRequest",
    "__abp"
  ]
}
```

PersonalInfoAPI.java class:

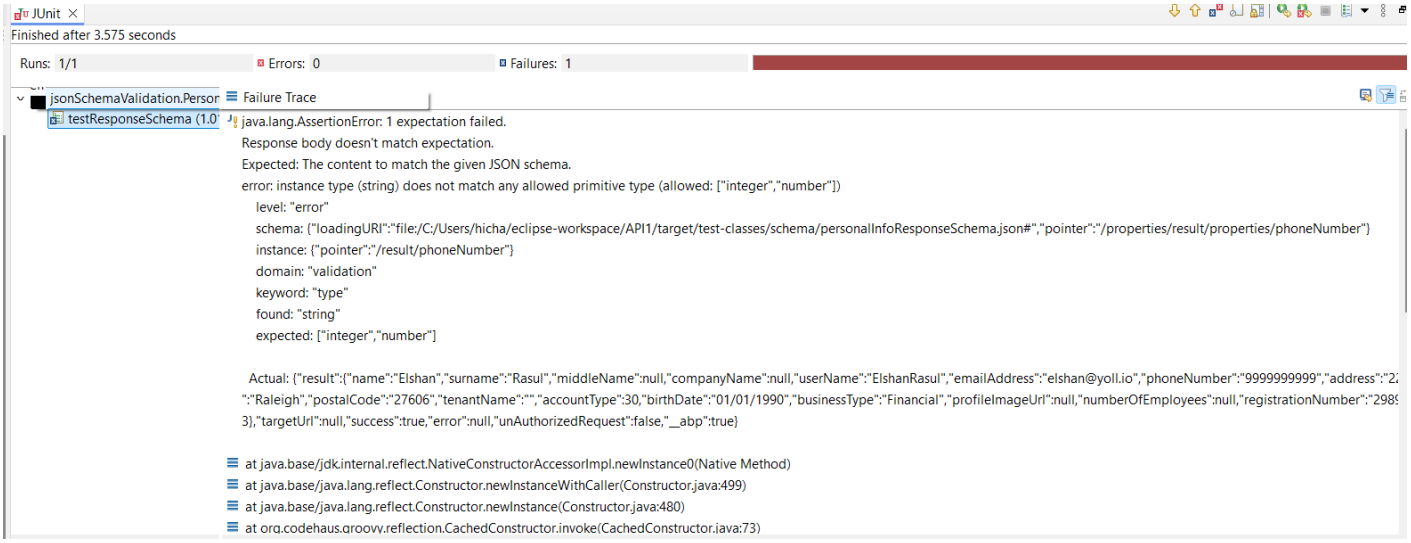



```
package jsonSchemaValidation;

import org.junit.Test;
import static io.restassured.RestAssured.given;
import static
io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchem
aInClasspath;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class PersonlaInfoAPI extends FunctionLibrary {
    @Test
    public void testResponseSchema() {
```

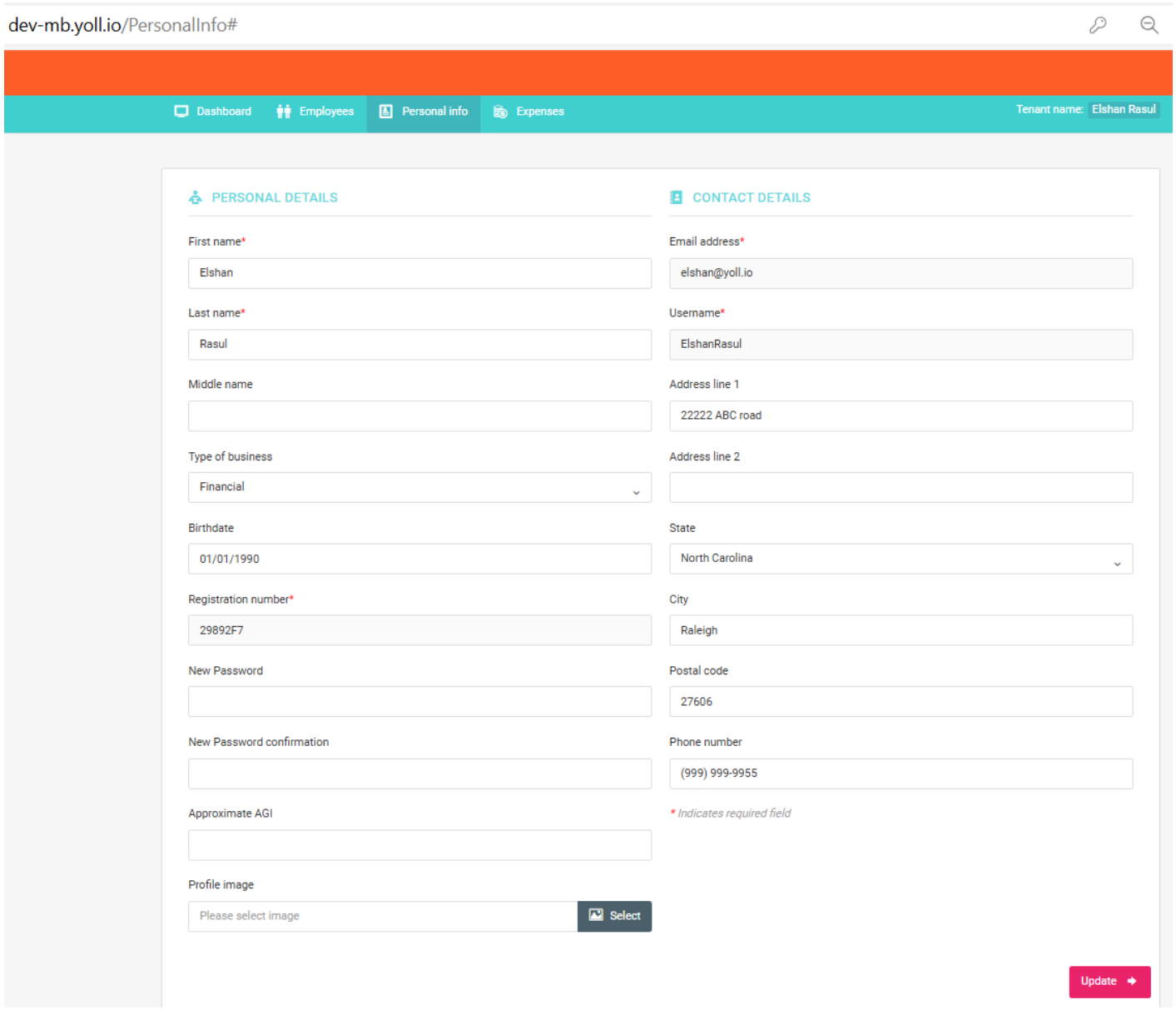
```
        String url =
"http://dev-mb.yoll.io/api/personalinfo";
        Response response = given().header("Authorization",
"Bearer " + getToken()).when().get(url);
        response.prettyPrint();


response.then().assertThat().body(matchesJsonSchemaInClasspath
("schema/personalInfoResponseSchema.json"));
    }
}
```

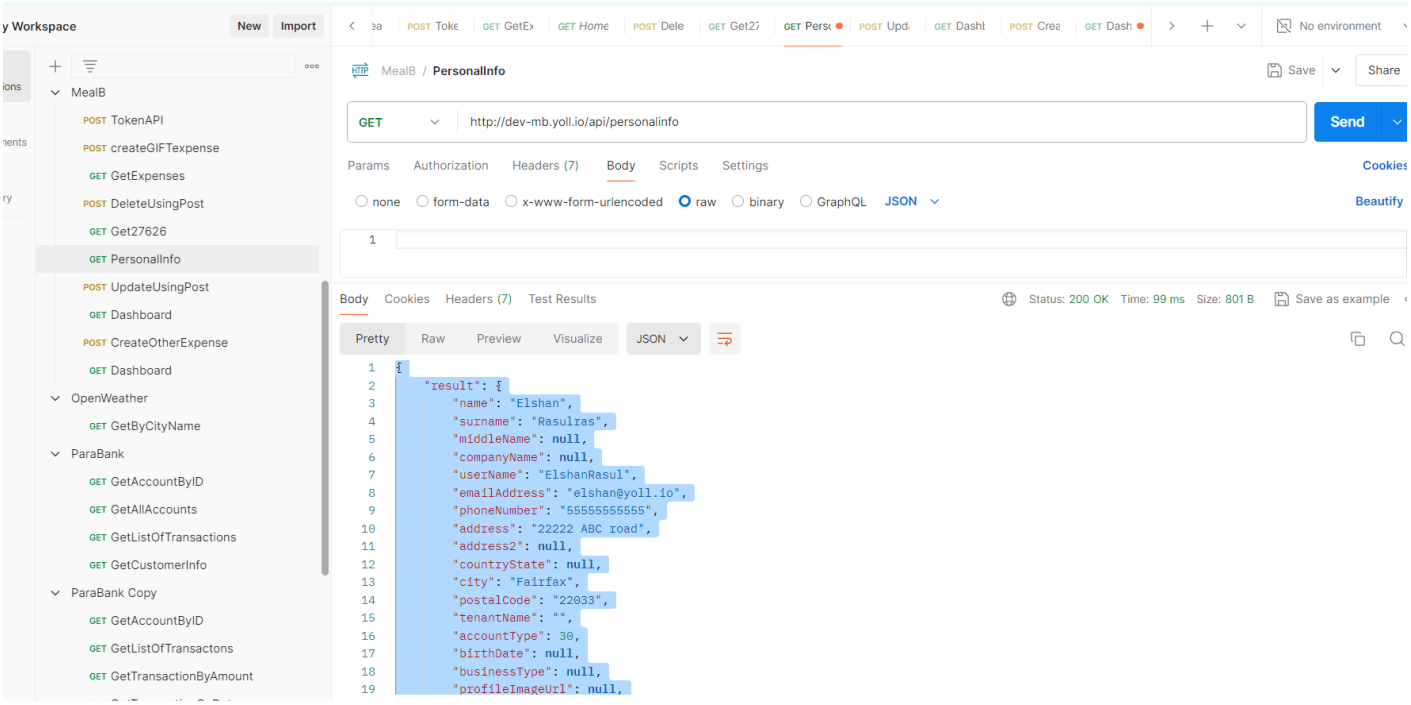
Result on the console:

```
java.lang.AssertionError: 1 expectation failed.

Response body doesn't match expectation.

Expected: The content to match the given JSON schema.

error: instance type (string) does not match any allowed primitive type
(allowed: ["integer","number"])

    level: "error"

    schema:
{"loadingURI":"file:/C:/Users/hicha/eclipse-workspace/API1/target/test-
classes/schema/personalInfoResponseSchema.json#","pointer":"/
properties/result/properties/phoneNumber"}

    instance: {"pointer":"/result/phoneNumber"}

    domain: "validation"

    keyword: "type"

    found: "string"

    expected: ["integer","number"]
```


JUnit
Finished after 3.575 seconds
Runs: 1/1
Errors: 0
Failures: 1
jsonSchemaValidation.Persor
testResponseSchema (1.0
Failure Trace
java.lang.AssertionError: 1 expectation failed.
Response body doesn't match expectation.
Expected: The content to match the given JSON schema.
error: instance type (string) does not match any allowed primitive type (allowed: ["integer","number"])
level: "error"
schema: {"loadingURI":"file:/C:/Users/hicha/eclipse-workspace/API1/target/test-classes/schema/personalInfoResponseSchema.json#","pointer":"/properties/result/properties/phoneNumber"}
instance: {"pointer":"/result/phoneNumber"}
domain: "validation"
keyword: "type"
found: "string"
expected: ["integer","number"]
Actual: {"result":{"name":"Elshan","surname":"Rasul","middleName":null,"companyName":null,"userName":"ElshanRasul","emailAddress":"elshan@yoll.io","phoneNumber":"9999999999","address":"2
":"Raleigh","postalCode":"27606","tenantName":"","accountType":30,"birthDate":"01/01/1990","businessType":"Financial","profileImageUrl":null,"numberOfEmployees":null,"registrationNumber":"298
3},"targetUrl":null,"success":true,"error":null,"unAuthorizedRequest":false,"__abp":true}
at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)
at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:480)
at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:73)

PersonalInfo on MEALB Website:





Result after converting json data :

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "surname": {
          "type": "string"
        },
        "middleName": {
          "type": "null"
        },
        "companyName": {
          "type": "null"
        },
```

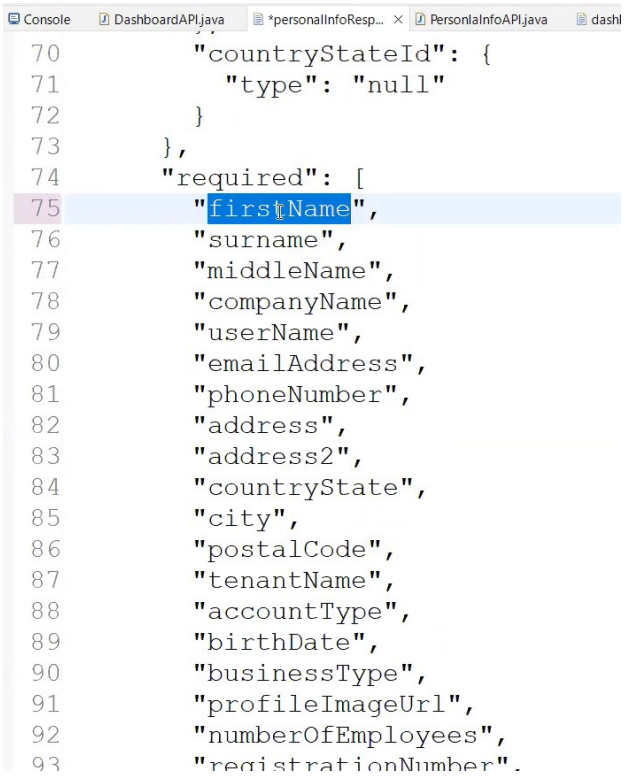
```
      "userName": {
        "type": "string"
      },
      "emailAddress": {
        "type": "string"
      },
        "phoneNumber": {
          "type": "string"
      },
      "address": {
        "type": "string"
      },
      "address2": {
        "type": "null"
      },
      "countryState": {
        "type": "string"
      },
      "city": {
        "type": "string"
      },
      "postalCode": {
        "type": "string"
      },
      "tenantName": {
        "type": "string"
      },
      "accountType": {
        "type": "integer"
      },
      "birthDate": {
        "type": "string"
      },
      "businessType": {
        "type": "string"
```

```
      },
      "profileImageUrl": {
        "type": "null"
      },
      "numberOfEmployees": {
        "type": "null"
      },
      "registrationNumber": {
        "type": "string"
      },
      "approximateAGI": {
        "type": "null"
      },
      "businessTypeId": {
        "type": "integer"
      },
      "countryStateId": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "surname",
      "middleName",
      "companyName",
      "userName",
      "emailAddress",
      "phoneNumber",
      "address",
      "address2",
      "countryState",
      "city",
      "postalCode",
      "tenantName",
      "accountType",
```

```
        "birthDate",
        "businessType",
        "profileImageUrl",
        "numberOfEmployees",
        "registrationNumber",
        "approximateAGI",
        "businessTypeId",
        "countryStateId"
      ]
    },
    "targetUrl": {
      "type": "null"
    },
    "success": {
      "type": "boolean"
    },
    "error": {
      "type": "null"
    },
    "unAuthorizedRequest": {
      "type": "boolean"
    },
    "__abp": {
      "type": "boolean"
    }
  },
  "required": [
    "result",
    "targetUrl",
    "success",
    "error",
    "unAuthorizedRequest",
    "__abp"
  ]
}
```

Another Example of error:

Validate the json file to see the result of the firstName. It supposed to be on required field on json file, But it does not exist on Response resul which will cause an error.

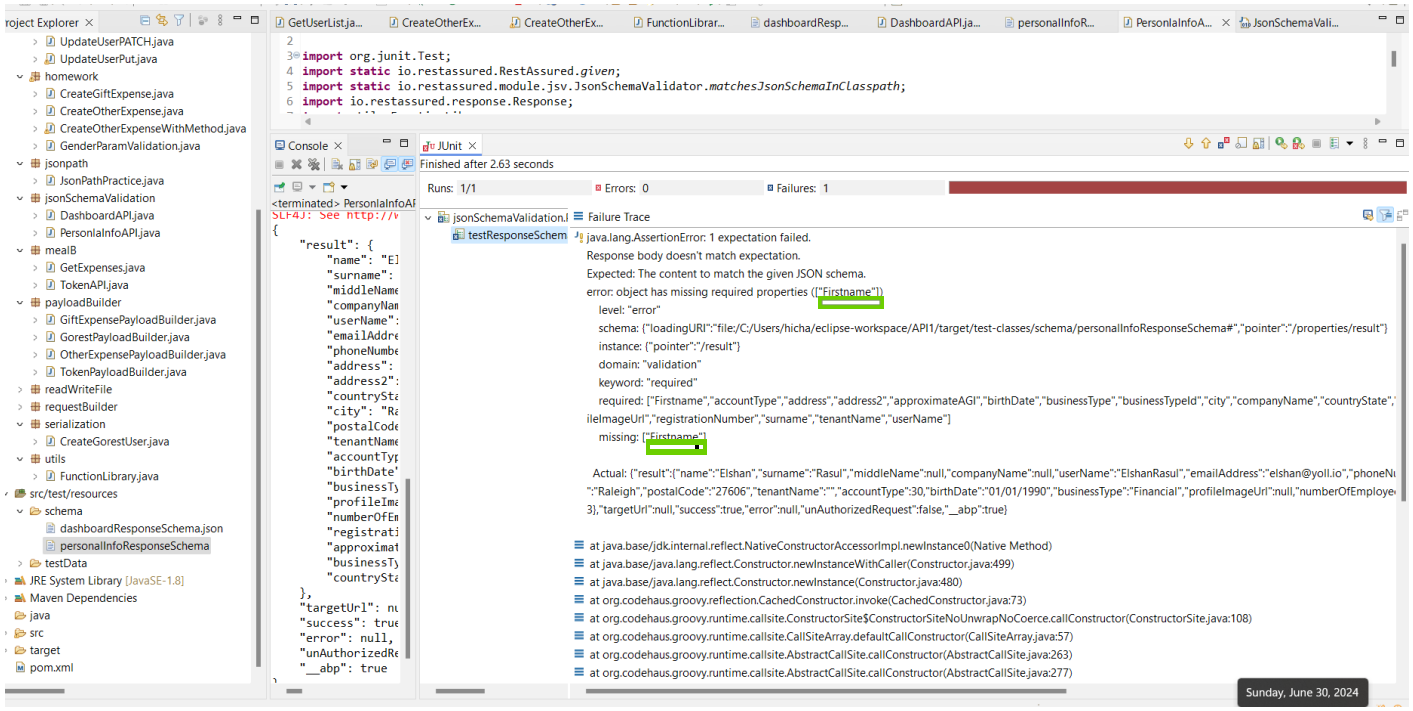



```
"required": [
        "firstName",
        "surname",
        "middleName",
        "companyName",
```

After running PersonalInfoAPI.java, we got this result:

```
java.lang.AssertionError: 1 expectation failed.
Response body doesn't match expectation.
Expected: The content to match the given JSON schema.
error: object has missing required properties (["firstName"])
    level: "error"
```

```
    schema:
{"loadingURI":"file:/C:/Users/hicha/eclipse-workspace/API1/target/test-
classes/schema/personalInfoResponseSchema.json#","pointer":"/
properties/result"}
    instance: {"pointer":"/result"}
    domain: "validation"
    keyword: "required"
    required:
["accountType","address","address2","approximateAGI","birthDate","busine
ssType","businessTypeId","city","companyName","countryState","countrySta
teId","emailAddress","firstName","middleName","numberOfEmployees","phone
Number","postalCode","profileImageUrl","registrationNumber","surname","t
enantName","userName"]
    missing: ["firstName"]
```



Dashboard API Calculation scenario:

1. Submit GET request to MealB Dashboard API
2. Retrieve the value of otherExpenses, travelExpenses, giftExpenses from response
3. Submit GET request to GetExpenses API
4. From response of GetExpenses API, get the total amount of all otherExpenses and validate against the value from Dashboard API
5. From response of GetExpenses API, get the total amount of all travelExpenses and validate against the value from Dashboard API
6. From response of GetExpenses API, get the total amount of all giftExpenses and validate against the value from Dashboard API

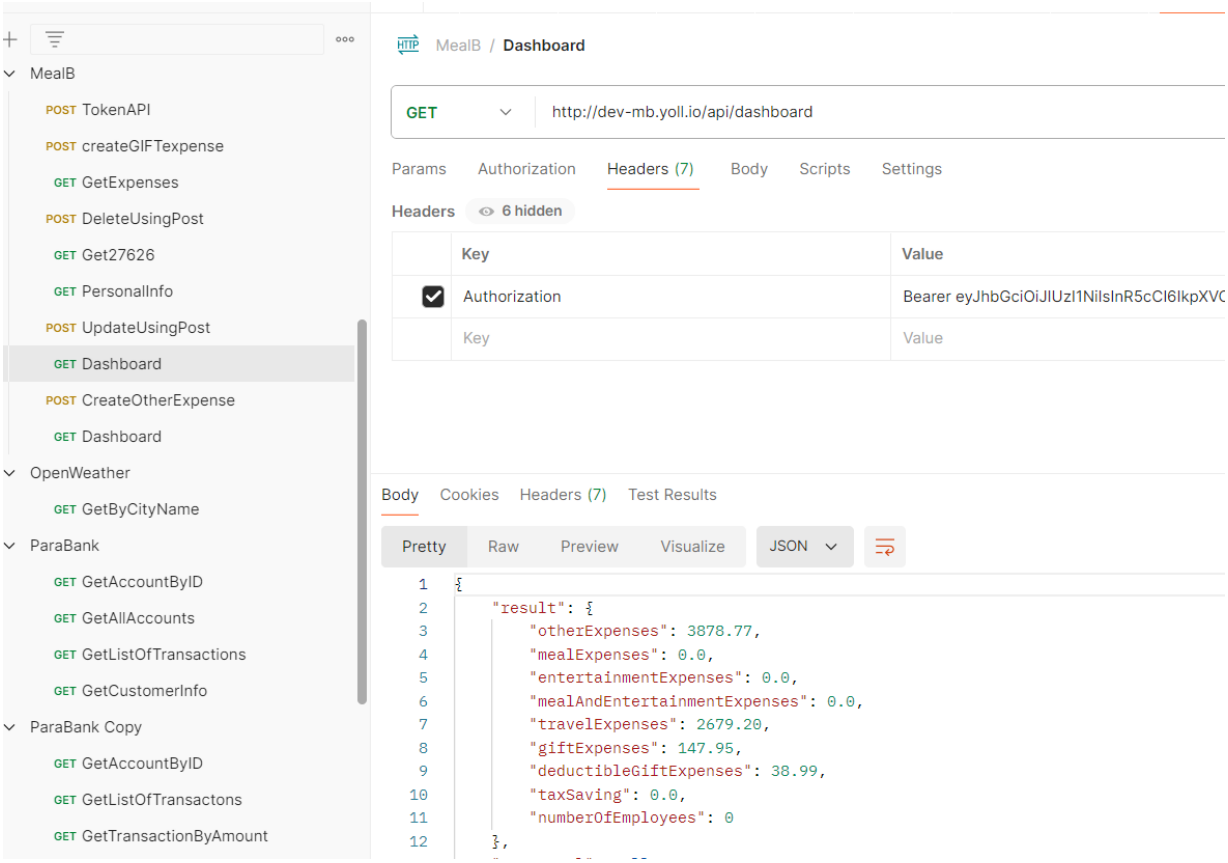
Note:

In GetExpense API response, where you see :

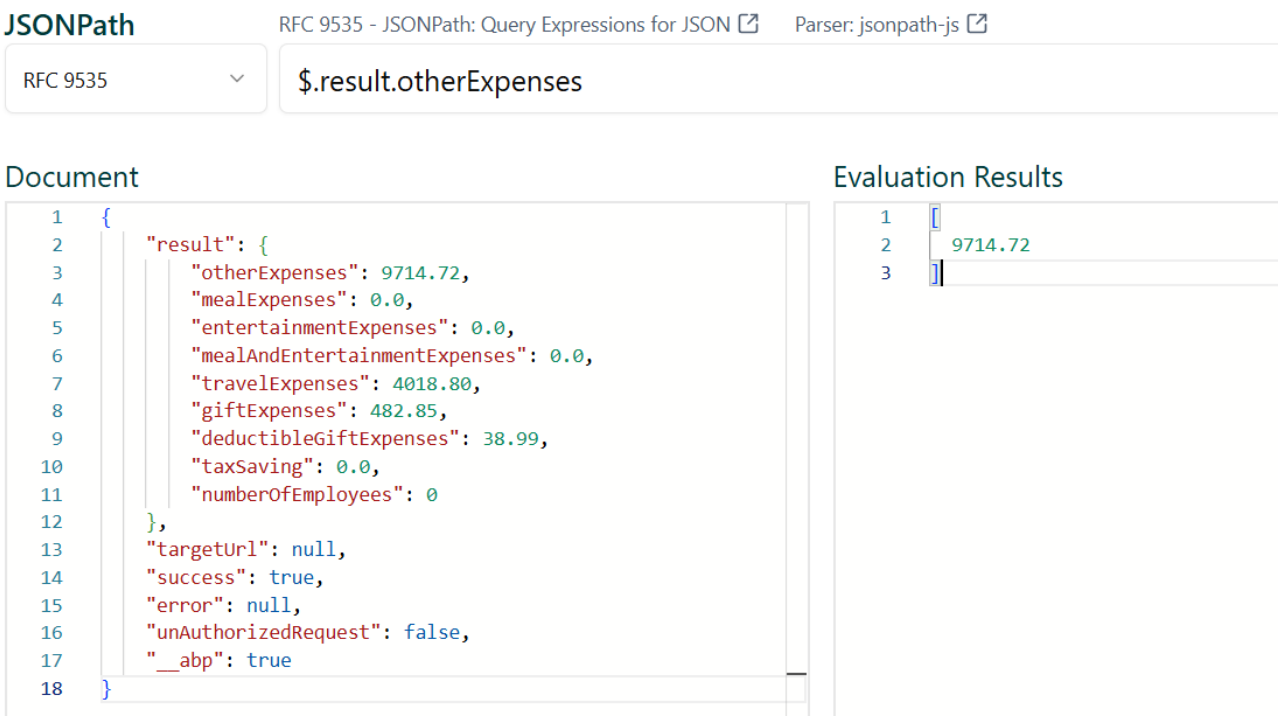
"expenseType": 10 >>> It means this is Other expense
"expenseType": 40 >>> It means this is Travel expense
"expenseType": 50 >>> It means this is Gift expense

☐ **Copy the result of Get Method for Dashboard (otherExpenses,travelExpenses,giftExpenses) on JSONPath Online Evaluator :**
https://jsonpath.com/

- **For DashBoard, Use Get Method of this url:** http://dev-mb.yoll.io/api/dashboard



☐ otherExpenses



☐ travelExpenses

JSONPath RFC 9535 - JSONPath: Query Expressions for JSON Parser: jsonpath-js

RFC 9535 $.result.travelExpenses

Document


```
{
  "result": {
    "otherExpenses": 9714.72,
    "mealExpenses": 0.0,
    "entertainmentExpenses": 0.0,
    "mealAndEntertainmentExpenses": 0.0,
    "travelExpenses": 4018.80,
    "giftExpenses": 482.85,
    "deductibleGiftExpenses": 38.99,
    "taxSaving": 0.0,
    "numberOfEmployees": 0
  },
  "targetUrl": null,
  "success": true,
  "error": null,
  "unAuthorizedRequest": false,
  "__abp": true
}
```


Evaluation Results

```
[
  4018.8
]
```

☐giftExpenses

JSONPath RFC 9535 - JSONPath: Query Expressions for JSON Parser: jsonpath-js

RFC 9535 $.result.giftExpenses

Document


```
{
  "result": {
    "otherExpenses": 9714.72,
    "mealExpenses": 0.0,
    "entertainmentExpenses": 0.0,
    "mealAndEntertainmentExpenses": 0.0,
    "travelExpenses": 4018.80,
    "giftExpenses": 482.85,
    "deductibleGiftExpenses": 38.99,
    "taxSaving": 0.0,
    "numberOfEmployees": 0
  },
  "targetUrl": null,
  "success": true,
  "error": null,
  "unAuthorizedRequest": false,
  "__abp": true
}
```

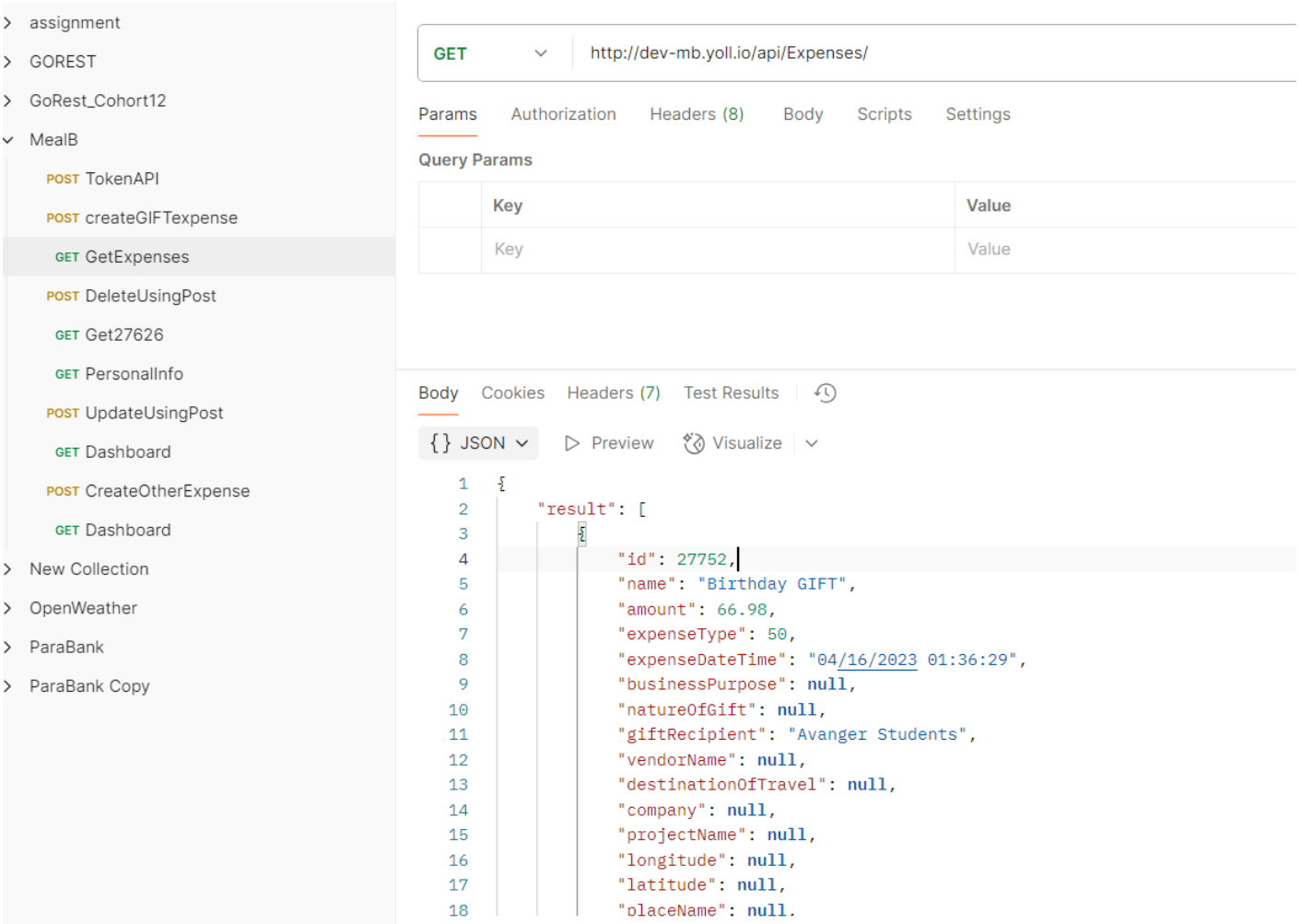

Evaluation Results

```
[
  482.85
]
```

- **Copy the result of Get Method for Expenses Types**

10 = otherExpenses,
40 = travelExpenses,
50 = giftExpenses on JSONPath Online Evaluator : https://jsonpath.com/

- For Expenses, Use **Get Method** of this url: http://dev-mb.yoll.io/api/Expenses



"expenseType": 10 >>> It means this is Other expense

JSONPath RFC 9535 - JSONPath: Query Expressions for JSON Parser: jsonpath-js

RFC 9535

```
$.result[?(@.expenseType==10)].amount
```

Document

```
"travelExpenseTypeId": null,
"otherExpenseNameId": 1,
"otherRelationship": null,
"receiptFile": null,
"receiptUrl": null
},
{
"id": 27730,
"name": "Electricity",
"amount": 555.50,
"expenseType": 10,
"expenseDateTime": "09/01/2020 01:36:29",
"businessPurpose": null,
"natureOfGift": null,
"giftRecipient": null,
"vendorName": null,
"destinationOfTravel": null,
"company": null,
"projectName": null,
"longitude": null,
"latitude": null,
"placeName": null,
"placeAddress": null,
"placeId": null,
"placeIcon": null,
```

Evaluation Results

```
[
444.99,
500,
555.5,
500,
555.5,
500,
500,
500,
444.99,
444.99,
444.99,
444.99,
99,
99,
10,
22.95,
22.95,
7,
22.95,
7,
7,
7,
7,
7,
444.99
```

"expenseType": 40 >>> It means this is Travel expense

JSONPath RFC 9535 - JSONPath: Query Expressions for JSON Parser: jsonpath-js

RFC 9535

```
$.result[?(@.expenseType==40)].amount
```

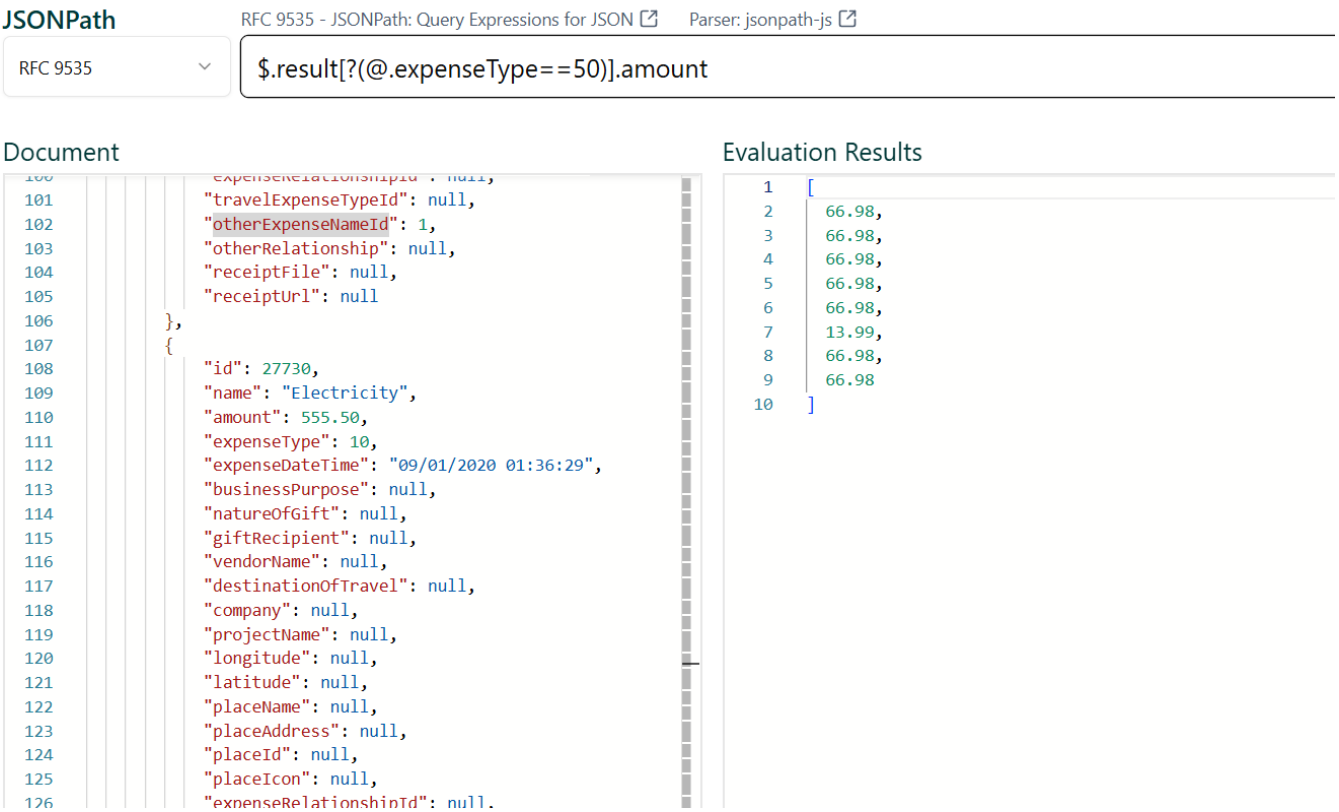
Document

```
"travelExpenseTypeId": null,
"otherExpenseNameId": 1,
"otherRelationship": null,
"receiptFile": null,
"receiptUrl": null
},
{
"id": 27730,
"name": "Electricity",
"amount": 555.50,
"expenseType": 10,
"expenseDateTime": "09/01/2020 01:36:29",
"businessPurpose": null,
"natureOfGift": null,
"giftRecipient": null,
"vendorName": null,
"destinationOfTravel": null,
"company": null,
"projectName": null,
"longitude": null,
"latitude": null,
"placeName": null,
"placeAddress": null,
"placeId": null,
```

Evaluation Results

```
[
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
66.98,
```

"expenseType": 50 >>> It means this is Gift expense



USING For Loop th calculate the total of each Type:

After using Json path for the three types of expenses, we got the expense of each type, then we need to calculate the total by using the for loop, then:

Print the total From Expenses.
Print the total From Dashboard.
Validation by comparing the total from Expenses and Dashboard.

Java Classes used:

Use FunctionLibrary Class.
Use DashboardAPICalculation extended from FunctionLibrary

Difference between double and Double in java

Use **double** for numerical computations and high-performance code. `double` is one of Java's eight primitive data types.

Use **Double** when working with APIs that require objects, such as Java Collections Framework.

🔹 Default Value:

- The default value of an uninitialized `double` variable is `0.0`.
- The default value of an uninitialized `Double` variable is `null`.

`assertEquals()` goes only with Double not double.

Round UP With : 0.01

```
assertEquals(travelExpenses, totalTravelExpense, 0.01);
```

Why Round Up in `assertEquals(otherExpenses, totalOtherExpense, 0.01);`?

The third parameter (`0.01`) in `assertEquals(expected, actual, delta)` defines a **tolerance range** when comparing two floating-point numbers.

⚠ Issue with Floating-Point Arithmetic

- **Double values in Java can have precision errors** due to how computers represent decimal numbers in binary.
- Example:

  ```
  System.out.println(0.1 + 0.2); // Output: 0.30000000000000004
  ```

  - Instead of `0.3`, it returns `0.30000000000000004`, causing **comparison failures**.

Rounding up (or setting a tolerance) prevents false failures due to floating-point precision errors**. The `0.01` tolerance ensures that small decimal variations** don't cause test failures unnecessarily**.**

FunctionLibrary Class:

```
package utils;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
```

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.TokenPayloadBuilder;

public class FunctionLibrary {
    String tokenUrl =
"http://dev-mb.yoll.io/api/tokenauth/authenticate";

    /**
     * @author ElshanRasulov This method submits POST request
to create token and
     *         retrieve token from response and return
     */
    public String getToken() {
        Response response =
given().body(createTokenPayload()).contentType(ContentType.JSO
N).when().post(tokenUrl);
        response.prettyPrint();
        assertEquals(200, response.getStatusCode());
//extract "accessToken" from response
        String token = JsonPath.read(response.asString(),
"$.result.accessToken");
        return token;
    }

    /**
     * @author ElshanRasulov This method create token request
payload data and
     *         convert it to string and return
     */
    public String createTokenPayload() {
        TokenPayloadBuilder payload = new
TokenPayloadBuilder();
        payload.setUsernameOrEmailAddress("ElshanRasul");
        payload.setPassword("Elshan123");
        return convertObjToString(payload);
    }

    /**
```

```
     * @author ElshanRasulov This method convert payload
objects to String format
     */
    public String convertObjToString(Object obj) {
        String strPayload = null;
        try {
            ObjectMapper objMap = new ObjectMapper();
            strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(obj
);
            System.out.println(strPayload);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return strPayload;
    }
}
```

DashboardAPICalculation Class extended from FunctionLibrary :

```
package homework;

import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import java.util.List;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class DashboardAPICalculation extends FunctionLibrary {
    String dashboardURL =
"http://dev-mb.yoll.io/api/dashboard";
    String getExpenseURL =
"http://dev-mb.yoll.io/api/Expenses";

    @Test
    public void homework() {

// 1. Submit GET request to MealB Dashboard API
```

```
        Response dashboardResp =
given().header("Authorization", "Bearer " +
getToken()).when().get(dashboardURL);
        dashboardResp.prettyPrint();

// 2. Retrieve the value of otherExpenses, travelExpenses,
giftExpenses from response
        Double otherExpenses =
JsonPath.read(dashboardResp.asString(),
"$.result.otherExpenses");
        Double travelExpenses =
JsonPath.read(dashboardResp.asString(),
"$.result.travelExpenses");
        Double giftExpenses =
JsonPath.read(dashboardResp.asString(),
"$.result.giftExpenses");
        System.out.println(otherExpenses);
        System.out.println(travelExpenses);
        System.out.println(giftExpenses);

// 3. Submit GET request to GetExpenses API
        Response expenseResp =
given().header("Authorization", "Bearer " +
getToken()).when().get(getExpenseURL);

// We do not need to print the Get Request because it has a
very long data
//expenseResp.prettyPrint();

// 4. From response of GetExpenses API, get the total amount
of all otherExpenses, and validate against the value from
Dashboard API

//"expenseType": 10 >>> It means this is Other expense

        List<Double> otherExpenseAmounts =
JsonPath.read(expenseResp.asString(),
                "$.result[?(@.expenseType==10)].amount");
        System.out.println(otherExpenseAmounts);
        Double totalOtherExpense = 0.0;
        for (Double eachValue : otherExpenseAmounts) {
            totalOtherExpense += eachValue;
        }
```

```
        // Print the total From Expenses:
        System.out.println("otherExpenseAmounts: " +
totalOtherExpense);
        // Print the total From Dashboard:
        System.out.println("dashboardAPIAmounts: " +
otherExpenses);
        //Validation by comparing the total from Expenses and
Dashboard:
        assertEquals(otherExpenses, totalOtherExpense, 0.01);

// 5. From response of GetExpenses API, get the total amount
of all travelExpenses, and validate against the value from
Dashboard API

//"expenseType": 40 >>> It means this is Travel expense

        List<Double> travelExpensesAmounts =
JsonPath.read(expenseResp.asString(),
                "$.result[?(@.expenseType==40)].amount");
        System.out.println(travelExpensesAmounts);

        Double totalTravelExpense = 0.0;

        for (Double eachValue : travelExpensesAmounts) {
            totalTravelExpense += eachValue;
        }
        System.out.println(totalTravelExpense);
        assertEquals(travelExpenses, totalTravelExpense,
0.01);

// 6. From response of GetExpenses API, get the total amount
of all giftExpenses, and validate against the value from
Dashboard API

//"expenseType": 50 >>> It means this is Gift expense

        List<Double> giftExpensesAmounts =
JsonPath.read(expenseResp.asString(),
                "$.result[?(@.expenseType==50)].amount");
        System.out.println(giftExpensesAmounts);

        Double totalGiftExpense = 0.0;
```

```
            for (Double eachValue : giftExpensesAmounts) {
                totalGiftExpense += eachValue;
            }

            System.out.println(totalGiftExpense);
            assertEquals(giftExpenses, totalGiftExpense, 0.01);
        }
}
```

RESULT :

```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "
",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
```

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

1. Submit GET request to MealB Dashboard API


```
{
    "result": {
        "otherExpenses": 3878.77,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 2679.20,
        "giftExpenses": 147.95,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


// 2. Retrieve the value of otherExpenses, travelExpenses, giftExpenses from response

3878.77
2679.2
147.95


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken": "
+Tam0uE23Suw85W9G8cw8zCYMzK8QYo/k7QKwWornd7b7F",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
```

```
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


```
// 3. Submit GET request to GetExpenses API

// 4. From response of GetExpenses API, get the total amount of all otherExpenses, and validate against the value from Dashboard API


[99.0,99.0,10.0,22.95,22.95,7.0,22.95,7.0,7.0,7.0,7.0,7.0,444.99,444.99,444.99,444.99,444.99,444.99,444.99,444.99]
otherExpenseAmounts: 3878.7699999999995
dashboardAPIAmounts: 3878.77

// 5. From response of GetExpenses API, get the total amount of all travelExpenses, and validate against the value from Dashboard API


[66.98,66.98,66.98,
```

1 Package and Imports

```
package homework;
```

- This defines the package name as `homework`.
- This is useful for **organizing the Java project** and helps avoid naming conflicts.

```
import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import java.util.List;
import io.restassured.response.Response;
import utils.FunctionLibrary;
```

- `import org.junit.Test;` → Allows writing unit tests using **JUnit**.
- `import com.jayway.jsonpath.JsonPath;` → Used to extract **specific values from JSON responses** using JSONPath.
- `import static io.restassured.RestAssured.given;` → `RestAssured` library is used to make **REST API requests**.
- `import static org.junit.Assert.assertEquals;` → Used for **asserting expected vs. actual values**.
- `import io.restassured.response.Response;` → Stores the HTTP response.
- `import utils.FunctionLibrary;` → This class extends `FunctionLibrary`, meaning it inherits helper methods, including `getToken()`.

---

2 Class Definition

```
public class DashboardAPICalculation extends FunctionLibrary {
```

- This class **inherits** (`extends`) `FunctionLibrary`, which contains reusable functions (like `getToken()`).
- This helps avoid **code duplication**.

---

3 Declaring API Endpoints

```
String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
String getExpenseURL = "http://dev-mb.yoll.io/api/Expenses";
```

- These store the **URLs for the APIs** we will be calling:
  - `dashboardURL`: Fetches the dashboard expenses summary.
  - `getExpenseURL`: Fetches the detailed list of expenses.

---

4 Test Method - `homework()`

The `@Test` annotation **marks this as a test case** that will be executed by JUnit.

```
@Test
public void homework() {
```

Step 1: Call Dashboard API and Print Response

```
Response dashboardResp = given().header("Authorization", "Bearer " +
getToken()).when().get(dashboardURL);
dashboardResp.prettyPrint();
```

- Calls the `dashboardURL` API using **GET request**.
- `getToken()` (from `FunctionLibrary`) retrieves the authentication token.
- `prettyPrint()` **prints the response** for debugging.

Step 2: Extract Expenses from Dashboard API

```
Double otherExpenses = JsonPath.read(dashboardResp.asString(),
"$.result.otherExpenses");
Double travelExpenses = JsonPath.read(dashboardResp.asString(),
"$.result.travelExpenses");
Double giftExpenses = JsonPath.read(dashboardResp.asString(),
"$.result.giftExpenses");
System.out.println(otherExpenses);
System.out.println(travelExpenses);
System.out.println(giftExpenses);
```

- Extracts `otherExpenses`, `travelExpenses`, and `giftExpenses` **from the JSON response** using JSONPath.

- `JsonPath.read(response, "$.result.otherExpenses")` → Reads the value from the `"result"` object in JSON.
- **Example JSON Response:**

```
{
  "result": {
    "otherExpenses": 100.50,
    "travelExpenses": 250.75,
    "giftExpenses": 50.00
  }
}
```

- These extracted values are printed for verification.

Step 3: Call GetExpenses API

```
Response expenseResp = given().header("Authorization", "Bearer " +
getToken()).when().get(getExpenseURL);
```

- Fetches **detailed expense records** from the `getExpenseURL` API.

Step 4: Validate `otherExpenses`

```
List<Double> otherExpenseAmounts = JsonPath.read(expenseResp.asString(),
"$.result[?(@.expenseType==10)].amount");
System.out.println(otherExpenseAmounts);
Double totalOtherExpense = 0.0;
for (Double eachValue : otherExpenseAmounts) {
    totalOtherExpense += eachValue;
}
System.out.println("otherExpenseAmounts: " + totalOtherExpense);
System.out.println("dashboardAPIAmounts: " + otherExpenses);
assertEquals(otherExpenses, totalOtherExpense, 0.01);
```

- Extracts **all expense amounts where `expenseType == 10`** (which represents `"Other Expenses"`).
- **Example Expense JSON:**

```
{
  "result": [
    { "expenseType": 10, "amount": 30.00 },
    { "expenseType": 10, "amount": 50.00 },
```

```
        { "expenseType": 10, "amount": 20.50 }
      ]
    }
```

- These values are summed (`totalOtherExpense`) and compared with `otherExpenses` from `Dashboard API`.
- The `assertEquals()` method checks if both values match **within a tolerance of 0.01**.

Step 5: Validate `travelExpenses`

```
List<Double> travelExpensesAmounts =
JsonPath.read(expenseResp.asString(), "$.result[?
(@.expenseType==40)].amount");
System.out.println(travelExpensesAmounts);
Double totalTravelExpense = 0.0;
for (Double eachValue : travelExpensesAmounts) {
    totalTravelExpense += eachValue;
}
System.out.println(totalTravelExpense);
assertEquals(travelExpenses, totalTravelExpense, 0.01);
```

- Extracts **all expenses where `expenseType == 40`** (which represents `"Travel Expenses"`).
- Summarizes and **validates it against Dashboard API**.

Step 6: Validate `giftExpenses`

```
List<Double> giftExpensesAmounts = JsonPath.read(expenseResp.asString(),
"$.result[?(@.expenseType==50)].amount");
System.out.println(giftExpensesAmounts);
Double totalGiftExpense = 0.0;
for (Double eachValue : giftExpensesAmounts) {
    totalGiftExpense += eachValue;
}
System.out.println(totalGiftExpense);
assertEquals(giftExpenses, totalGiftExpense, 0.01);
```

- Extracts **all expenses where `expenseType == 50`** (which represents `"Gift Expenses"`).
- Summarizes and **validates it against Dashboard API**.

Summary

1. **Calls the Dashboard API** to retrieve summarized expenses.
2. **Extracts values** for `"otherExpenses"`, `"travelExpenses"`, and `"giftExpenses"`.
3. **Calls the GetExpenses API** to retrieve detailed expense records.
4. **Filters expenses by type** (`otherExpenses = 10`, `travelExpenses = 40`, `giftExpenses = 50`).
5. **Summarizes the expenses** retrieved from `GetExpenses API`.
6. **Validates if the summarized values match the Dashboard API**.

Deserialization

Deserialization is the process of converting JSON or XML responses into Java objects.

When an API returns a JSON response, you would deserialize this response to verify its contents, check for expected values, and validate data integrity.

How Deserialization Works in Java ?

- **Step 1:** API sends JSON response (as a `String`).

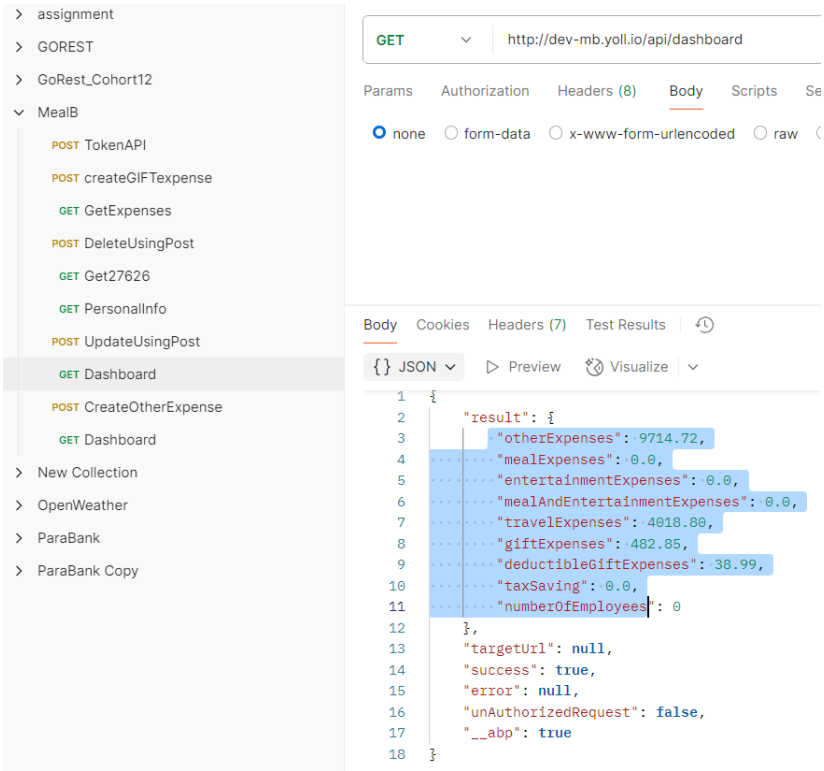
- **Step 2:** We use `ObjectMapper` (from the Jackson library) to **convert the JSON into a Java object**.
- **Step 3:** We access the object's properties using **getters** or direct references.

 ObjectMapper: Creates an instance of `ObjectMapper` from the Jackson library, which is used to map JSON strings to Java objects.

```
ObjectMapper objMap = new ObjectMapper();
DashboardResponse dashObj =
objMap.readValue(dashboardResp.asString(),DashboardResponse.class);
```

Deserialization: The response body (JSON) is converted to a `DashboardResponse` object using the `readValue` method of `ObjectMapper`.

For the Get Method of DASHBOARD, We need to use getters and setters from the Json data:



We will use 4 Classes :
EXPENSES and DashboardResponse For Getters.
TestDashboardAPIDeserialization Class extended from FunctionLibrary for deserialization

THE VARIABLES WITH GREEN COLOR ARE Children, AND SHOULD BE ON THE EXPENSES CLASS, THEN WE PUT THE ELEMENTS WITH YELLOW COLOR ON A DIFFERENT CLASS:DashboardResponse.

Both EXPENSES and DashboardResponse contain Getters and Setters.

Always put the chlidren in a separeted Class !


```
{
    "result": {
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


Pay attention to the variables types: double, int, String, boolean...

"numberOfEmployees": 0 is int

"taxSaving": 0.0, is double

"unAuthorizedRequest": false, is boolean

"error": null, is String

We need to assign "result" to Expenses on `DashboardResponse` Class

```
public class DashboardResponse {
	private Expenses result;
	private String targetUrl;
	private boolean success;
	private String error;
	private boolean unAuthorizedRequest;
	private boolean __abp;
```

This line of code is declaring a **private instance variable** named `result` of type `Expenses`.

Breaking it Down:

1. **`private`** → This means the variable can only be accessed within the same class.
2. **`Expenses`** → This is a **custom class** that represents the structure of the **`result`** field in a JSON response.
3. **`result`** → The variable name, which will store an instance of the **`Expenses`** **class**: **otherExpenses, mealExpenses, entertainmentExpenses, mealAndEntertainmentExpenses, travelExpenses, giftExpenses, deductibleGiftExpenses, taxSaving, numberOfEmployees**

Under deserialization package, create the Class Expenses for Getters and Setters:

```
package deserialization;

public class Expenses {
	private double otherExpenses;
	private double mealExpenses;
	private double entertainmentExpenses;
	private double mealAndEntertainmentExpenses;
	private double travelExpenses;
	private double giftExpenses;
	private double deductibleGiftExpenses;
	private double taxSaving;
	private int numberOfEmployees;

	public double getOtherExpenses() {
		return otherExpenses;
	}

	public void setOtherExpenses(double otherExpenses) {
		this.otherExpenses = otherExpenses;
	}

	public double getMealExpenses() {
		return mealExpenses;
```

```
	}

	public void setMealExpenses(double mealExpenses) {
		this.mealExpenses = mealExpenses;
	}

	public double getEntertainmentExpenses() {
		return entertainmentExpenses;
	}

	public void setEntertainmentExpenses(double entertainmentExpenses) {
		this.entertainmentExpenses = entertainmentExpenses;
	}

	public double getMealAndEntertainmentExpenses() {
		return mealAndEntertainmentExpenses;
	}

	public void setMealAndEntertainmentExpenses(double mealAndEntertainmentExpenses)
{
		this.mealAndEntertainmentExpenses = mealAndEntertainmentExpenses;
	}

	public double getTravelExpenses() {
		return travelExpenses;
	}

	public void setTravelExpenses(double travelExpenses) {
		this.travelExpenses = travelExpenses;
	}

	public double getGiftExpenses() {
		return giftExpenses;
	}

	public void setGiftExpenses(double giftExpenses) {
		this.giftExpenses = giftExpenses;
	}

	public double getDeductibleGiftExpenses() {
		return deductibleGiftExpenses;
	}

	public void setDeductibleGiftExpenses(double deductibleGiftExpenses) {
		this.deductibleGiftExpenses = deductibleGiftExpenses;
	}

	public double getTaxSaving() {
		return taxSaving;
	}

	public void setTaxSaving(double taxSaving) {
		this.taxSaving = taxSaving;
	}

	public int getNumberOfEmployees() {
		return numberOfEmployees;
	}

	public void setNumberOfEmployees(int numberOfEmployees) {
		this.numberOfEmployees = numberOfEmployees;
```

```
	}
}
```

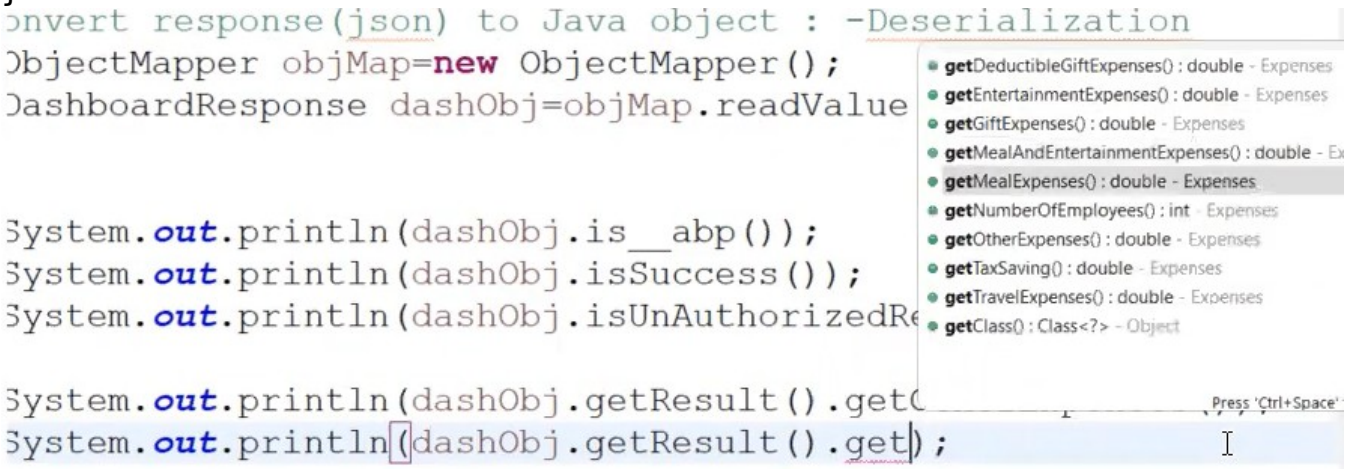
Getters and Setters of DashboardResponse Class

```
package deserialization;

public class DashboardResponse {
	private Expenses result;
	private String targetUrl;
	private boolean success;
	private String error;
	private boolean unAuthorizedRequest;
	private boolean __abp;

	public Expenses getResult() {
		return result;
	}

	public String getTargetUrl() {
		return targetUrl;
	}

	public boolean isSuccess() {
		return success;
	}

	public String getError() {
		return error;
	}

	public boolean isUnAuthorizedRequest() {
		return unAuthorizedRequest;
	}

	public boolean is__abp() {
		return __abp;
	}
}
```

DashboardResponse Class for deserialization:

```
package deserialization;

import static io.restassured.RestAssured.given;
import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.restassured.response.Response;
import utils.FunctionLibrary;
```

```java
public class TestDashboardAPIDeserialization extends FunctionLibrary {
     @Test
//Make sure it is http not https !
     public void test() throws JsonMappingException,
JsonProcessingException {
          String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
          Response dashboardResp = given().header("Authorization",
"Bearer " + getToken()).when().get(dashboardURL);
          dashboardResp.prettyPrint();
// convert response(json) to Java object : -Deserialization
          ObjectMapper objMap = new ObjectMapper();
          DashboardResponse dashObj =
objMap.readValue(dashboardResp.asString(), DashboardResponse.class);
// For boolean, the variable with is:

//public boolean isSuccess() {return success;}

//public boolean isUnAuthorizedRequest() {return unAuthorizedRequest;}

//public boolean is__abp() {return __abp;    }

// We can access to the Get methods directly
          System.out.println(dashObj.is__abp());
          System.out.println(dashObj.isSuccess());
          System.out.println(dashObj.isUnAuthorizedRequest());

// we need to use Getters to print the elements:
// We cannot access to the Get methods directly because they are
children. We use getResult() method first as a parent.
          System.out.println(dashObj.getResult().getOtherExpenses());
          System.out.println(dashObj.getResult().getMealExpenses());


System.out.println(dashObj.getResult().getMealAndEntertainmentExpenses()
);
     }
}
```



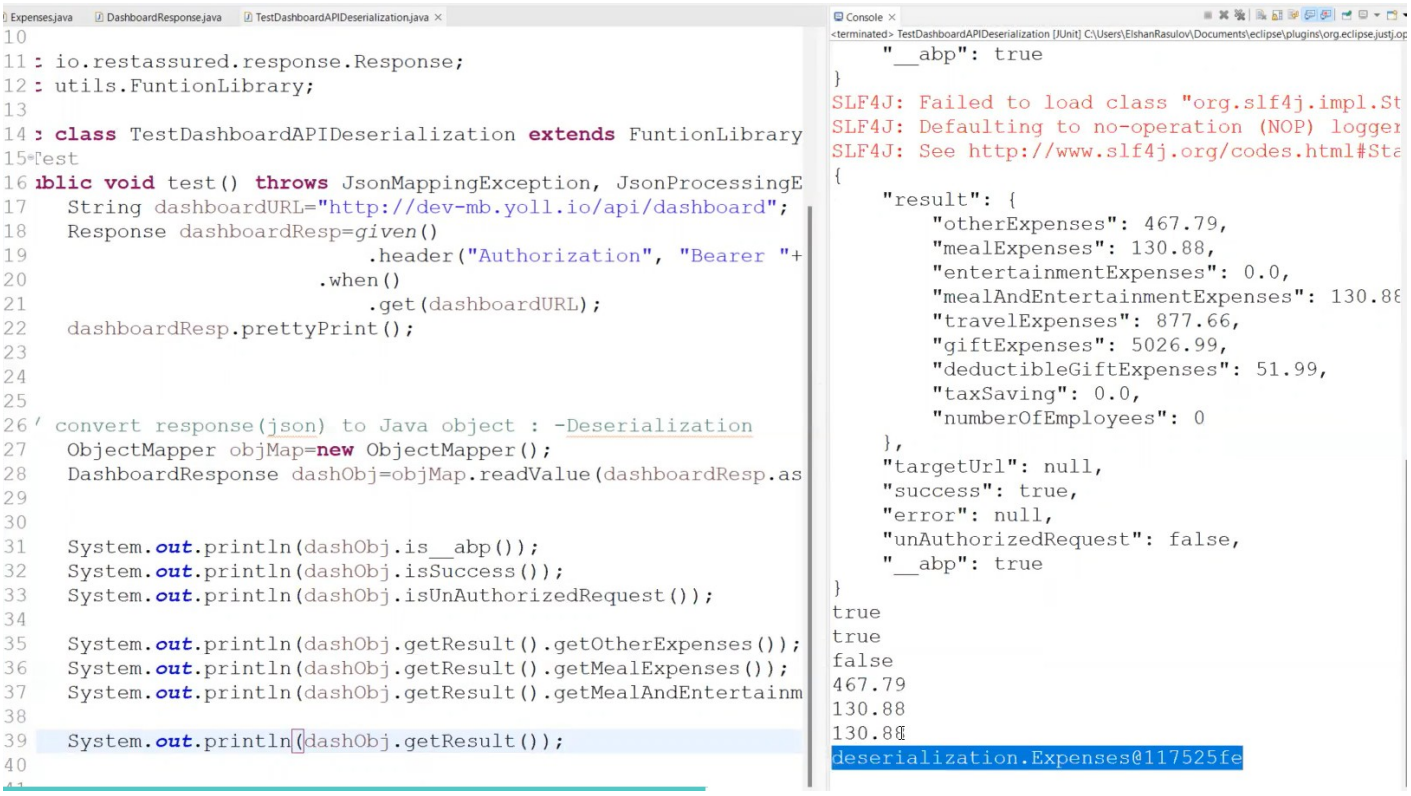
We cannot print the variables by using only: getResult()

```
System.out.println(dashObj.getResult());
```

The result will be: Deserialization.Expenses@117525fe

How can we use to get the result ?
We need to use the MAP.

See the second method of running TestDashboardAPIDeserialization by using MAP on the topic named: Deserialization using Map:



Result after running TestDashboardAPIDeserialization class:

```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
```

```
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiJmNTUwMGI2NC03MjE4LTRkZjQtODM5My03OWQ1NDc0NjI0ZjkiLCJ
pYXQiOjE3NDIxNDI1NjIsIm5iZiI6MTc0MjE0MjU2MiwiZXhwIjoxNzczNjc4NTYyLCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.b7sqJn6V7d7ET9QT-dTtD3dJYx2rRT1ieqFBiip__wg",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CR5E9kLumaSiHc9RYmjk1FLuPVm6nEduFgd3
1qQVF9T43u6Vekvj8VQTI3f6HuhgKOv3sOO5tuaqiawSjTPyCp9G04nl2X3Xvx6uxDSLMp96l5rFa+bL0S52qQV
7M8U1YsSBRwHYvbFy16IHAGtGTgvW7Ykh1El4YgH3BBQKteyu66v+nI92waYSAqjPvNgogygMOKm16G+byo5pF4
eIQLRJ8YU9AYDrfdUcjDnVAEiuVkqHUx71UYhGoBRcyuq9WZhQFAbC9kg3vjLSE/
HTKiCCFl7WMnUhHMLD8LkbcbQu",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
```

```
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
true
true
false
9714.72
0.0
0.0
```


Explanation of `TestDashboardAPIDeserialization` Class

This class tests the deserialization of a JSON response from an API into a Java object.

Step-by-Step Explanation

1 Import Statements

```
import static io.restassured.RestAssured.given;
import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.restassured.response.Response;
import utils.FunctionLibrary;
```

- **RestAssured**: Used to send API requests.
- **JUnit Test**: Used to mark the test method.
- **Jackson ObjectMapper**: Used for deserialization (converting JSON to Java objects).
- **Response**: Represents the API response.
- **FunctionLibrary**: Contains helper methods like `getToken()`.

---

2 Class Definition

```
public class TestDashboardAPIDeserialization extends FunctionLibrary {
```

- Extends `FunctionLibrary`, meaning it inherits methods from that class (e.g., `getToken()`).

---

3 Test Method

```
public void test() throws JsonMappingException, JsonProcessingException {
```

- **`@Test`**: Marks this method as a JUnit test.
- **Throws Clause**: Handles potential exceptions when converting JSON to a Java object.

---

4 Sending a GET Request

```
String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
Response dashboardResp = given()
    .header("Authorization", "Bearer " + getToken())
    .when()
    .get(dashboardURL);
dashboardResp.prettyPrint();
```

- **`given()`** → Starts building an API request.
- **`header("Authorization", "Bearer " + getToken())`** → Adds the authentication token to the request.
- **`when().get(dashboardURL)`** → Sends a **GET request** to the dashboard API.
- **`prettyPrint()`** → Prints the raw JSON response for debugging.

---

5 Deserialization: Converting JSON to Java Object

```
ObjectMapper objMap = new ObjectMapper();
DashboardResponse dashObj = objMap.readValue(dashboardResp.asString(),
DashboardResponse.class);
```

- **`ObjectMapper`**: A Jackson class used to map JSON into Java objects.
- **`readValue(dashboardResp.asString(), DashboardResponse.class)`**:
  - Converts the JSON response to a **DashboardResponse** Java object.
  - readValue() **parses JSON** and **maps** it to DashboardResponse.

`readValue()` **takes two arguments**:

- **The JSON string** → `dashboardResp.asString()`
- **The target Java class** → `DashboardResponse.class`

---

6 Accessing Deserialized Data

```
System.out.println(dashObj.is__abp());
System.out.println(dashObj.isSuccess());
System.out.println(dashObj.isUnAuthorizedRequest());
System.out.println(dashObj.getResult().getOtherExpenses());
System.out.println(dashObj.getResult().getMealExpenses());
System.out.println(dashObj.getResult().getMealAndEntertainmentExpenses());
```

- **Prints values from the `DashboardResponse` object**:
  - **Booleans** (`is__abp()`, `isSuccess()`, `isUnAuthorizedRequest()`)
  - **Expense values** (`getOtherExpenses()`, `getMealExpenses()`, etc.)

🔍 How Does Deserialization Work?

1. The **API returns a JSON response**.
2. **Jackson's `ObjectMapper` converts JSON into a Java object (`DashboardResponse`)**.
3. The **Java object fields are populated** with data from the API response.
4. The **program can access values** using **getter methods**.

Deserialization using Map:

```
private Map<String, Object> result;
```

Using **Map<String, Object>** makes the code **more flexible** for handling **dynamic JSON responses**. If the JSON structure is **fixed**, it is better to use a **strongly typed Java class** (Expenses). However, if the structure **can change**, using a Map prevents breaking changes.

In our Case, **result** contains Variables having different data types: String, double for:

OtherExpenses, MealExpenses, MealAndEntertainmentExpenses

We will use 3 Classes :
-DashboardResponseMap **For Getters.**
- TestDashboardAPIDeserialization Class **extended from FunctionLibrary for deserialization**

Getters: DashboardResponseMap

```java
package deserialization;
import java.util.Map;

public class DashboardResponseMap {

    private Map<String, Object> result;
    private String targetUrl;
    private boolean success;
    private String error;
    private boolean unAuthorizedRequest;
    private boolean __abp;

    public Map<String, Object> getResult() {
        return result;
    }

    public void setResult(Map<String, Object> result) {
        this.result = result;
    }

    public String getTargetUrl() {
        return targetUrl;
    }

    public void setTargetUrl(String targetUrl) {
        this.targetUrl = targetUrl;
    }

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public String getError() {
        return error;
    }

    public void setError(String error) {
        this.error = error;
    }

    public boolean isUnAuthorizedRequest() {
        return unAuthorizedRequest;
    }

    public void setUnAuthorizedRequest(boolean unAuthorizedRequest) {
        this.unAuthorizedRequest = unAuthorizedRequest;
```

```
    }

    public boolean is__abp() {
        return __abp;
    }

    public void set__abp(boolean __abp) {
        this.__abp = __abp;
    }
}
```

Explaination why we use Object instead of primitive types

In the `DashboardResponseMap` class, the `result` field is defined as:

```
private Map<String, Object> result;
```

Instead of using a **specific class** (`Expenses` with `double` fields) or **primitive types**, we are using a **Map** (`Map<String, Object>`).

Reason 1: Dynamic Structure Handling

- The `result` field in the JSON response is an **object**, but we **don't know the exact structure** at compile time.
- By using `Map<String, Object>`, we allow **flexibility** to handle **any kind of data dynamically**.

Example JSON Response

```
{
  "result": {
    "otherExpenses": 100.50,
    "mealExpenses": 200.75,
    "taxSaving": 50.25
  },
  "success": true,
  "unAuthorizedRequest": false,
  "__abp": true
}
```

- In this case, the `result` object has different fields (`otherExpenses`, `mealExpenses`, etc.), and we **don't want to hardcode them**.
- Instead, we **store them in a Map** where:

- The **keys** (e.g., `"otherExpenses"`, `"mealExpenses"`) are **strings**.
- The **values** (e.g., `100.50`, `200.75`) can be **any data type** (double, int, String, etc.).

Reason 2: Handling Unknown or Variable Keys

- Suppose the API response **changes** and adds a new field (`"newExpense"`) or removes one.
- If we use a **fixed class (Expenses)**, the code **will break** when the API changes.
- Using `Map<String, Object>` makes the code **more generic and adaptable**.

Example - Retrieving Values from the Map

```
System.out.println(result.get("otherExpenses"));  // 100.50
System.out.println(result.get("mealExpenses"));  // 200.75
System.out.println(result.get("taxSaving"));     // 50.25
```

- The values are **retrieved dynamically** without needing a predefined class.

Reason 3: Avoiding Strict Data Type Conversions

- If we use `double` or `int`, we **must ensure** that every value is always a number.
- But in **some cases**, API responses can be inconsistent:

```
{
  "result": {
    "otherExpenses": "100.50",   // String instead of a
number!
    "mealExpenses": 200.75
  }
}
```

- Using `Object` allows **flexibility** to store **any data type** (String, Number, Boolean, etc.), avoiding unnecessary conversion issues.

TestDashboardResponseMap Class for deserialization

```
package deserialization;

import static io.restassured.RestAssured.given;
import java.util.Set;
import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class TestDashboardResponseMap extends FunctionLibrary
{
     @Test
     public void test() throws JsonMappingException,
JsonProcessingException {
          String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
          Response dashboardResp = given().header("Authorization",
"Bearer " + getToken()).when().get(dashboardURL);
          dashboardResp.prettyPrint();
// convert response(json) to Java object : -Deserialization
          ObjectMapper objMap = new ObjectMapper();
          DashboardResponseMap dashObj =
objMap.readValue(dashboardResp.asString(), DashboardResponseMap.class);
          System.out.println(dashObj.is__abp());
          System.out.println(dashObj.isSuccess());
          System.out.println(dashObj.isUnAuthorizedRequest());
          System.out.println(dashObj.getResult());

}}
```

Result of Map in Yellow:


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiI2Njc0N2FhNi0wYjhlLTQ5NDUtOWQ4Yy1mZThkY2JmYjA5MGUiLCJ
pYXQiOjE3NDIxNTEwNTMsIm5iZiI6MTc0MjE1MTA1MywiZXhwIjoxNzczNjg3MDUzLCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.Xhlb8zbXqYwodeeZvBamEXfZlFn0-ShPYaBeClLLaXs",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
```

```
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CSaqLmy9sJI0/3atzuzvIEl/
YkYVCP88rcOIUfSjmroKS09cPlhEhZYKSj0nfLMa7PFWEqUGpeS8N96bLgsucgIxjUmIl7mwufZU5lgw2FaAzzK
nSy2dty7rA5Jo8gDyF2/
IJBoomTmuEDi+1y0FVTRN9I6djT6MaM1OUKjtrNSZLphDBtLC0tTLOZ24lPGbYuMnJv+UkZtWEf9hY4CAs0/
m32lKxDuxWxiXPxM3lppYXVpDOq/2aS4uxz35MIxF2GsMfZnRx6i1Oh/w9S/EVp9ie3bFLGq96STjdvc6xbFV",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
```

```
    "__abp": true
}
true
true
false
{otherExpenses=9714.72, mealExpenses=0.0, entertainmentExpenses=0.0,
mealAndEntertainmentExpenses=0.0, travelExpenses=4018.8, giftExpenses=482.85,
deductibleGiftExpenses=38.99, taxSaving=0.0, numberOfEmployees=0}
```

Using Set and KeySet() to retrive TestDashboardResponseMap Class for deserialization

```
package deserialization;

import static io.restassured.RestAssured.given;
import java.util.Set;
import org.junit.Test;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.restassured.response.Response;
import utils.FunctionLibrary;

public class TestDashboardResponseMap extends FunctionLibrary
{
     @Test
     public void test() throws JsonMappingException,
JsonProcessingException {
          String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
          Response dashboardResp = given().header("Authorization",
"Bearer " + getToken()).when().get(dashboardURL);
          dashboardResp.prettyPrint();

// convert response(json) to Java object : -Deserialization
          ObjectMapper objMap = new ObjectMapper();
          DashboardResponseMap dashObj =
objMap.readValue(dashboardResp.asString(), DashboardResponseMap.class);
          System.out.println(dashObj.is__abp());
          System.out.println(dashObj.isSuccess());
          System.out.println(dashObj.isUnAuthorizedRequest());
          System.out.println(dashObj.getResult());

      // Retrieving all keys: Extracted the keys using keySet()
  and stored them in a Set<String> keySet.


          Set<String> keySet = dashObj.getResult().keySet();

   // Iterating through keys and printing key-value pairs
```

```
            for (String key : keySet) {
                 System.out.println(key + ": " +
dashObj.getResult().get(key));
            }

              // Printing all keys
            System.out.println(dashObj.getResult().keySet());
            // Printing all values
            System.out.println(dashObj.getResult().values());
      }
}
```

Result on Console:


```
{
  "usernameOrEmailAddress" : "ElshanRasul",
  "password" : "Elshan123"
}
{
    "result": {
        "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1L
zA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEwNzgxIiwiaHR0cDovL3NjaGVtYXMueG1sc29h
cC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6IkVsc2hhblJhc3VsIiwiQXNwTmV0LklkZW5
0aXR5LlNlY3VyaXR5U3RhbXAiOiJkMjI1NTliNC05Yzk4LTQyOGItYjEyZS05YTNkYjI5ZDQ4MjMiLCJodHRwOi
8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTZWxmRW1wb
G95ZWQiLCJodHRwOi8vd3d3LmFzcG5ldGJvaWxlcnBsYXRlLmNvbS9pZGVudGl0eS9jbGFpbXMvdGVuYW50SWQi
OiIxIiwic3ViIjoiMTA3ODEiLCJqdGkiOiIwZGFmNWYxMS0zNzY5LTQ1YzYtOTM3Yi1hZDU5YmNmMzJkODQiLCJ
pYXQiOjE3NDIyMjcyMTAsIm5iZiI6MTc0MjIyNzIxMCwiZXhwIjoxNzczNzYzMjEwLCJpc3MiOiJNZWFsQiIsIm
F1ZCI6Ik1lYWxCIn0.1GpFPKgklMarXwYXUuijtHWHNFgNTicZArVUVQD4nBg",
        "encryptedAccessToken":
"wNYmO41/48SHNstaLVXxHCCre29BZQl1NhC6NM3R3rzpXtPQxVzH6jEzA/QhXFN5tu6Fk7pO53uppm1mVXMZgx
byRVz26dnepi/FyB6axBY+6gq1GL+uRQgoiFUCjRN2p8w6LevViwKlHyWZZJZO1DGVSjAi1m2U+og9pkHw9/
RKwrnVrK15zItPBZE8KRZXCeYekIizZP4xZfH22A6SKolN287gNUwkLJ/
QL0s70KC6ZS4JJ+siJqELz5n2/0fY8+OcxxeL7hbFeYdUzwA6lIGdMYDX+yXDE1ZysN7eIRMoQBnk1nVV0N9Ojn
wb1btHDkWRXCLa8PJUyut2QTqsXI563AjRRPMYKJyZWzKXkPiXP7GxEUBhMhNq9iFspbtCLpZjCWXJoEX9yGTE8
WvJE6bHmCSnZiEwuV9giCKDmvVVZtqsqW+gdsjl0e+cOMcAGsN4giTNiba3Qx5wySLEUNDHNXncBVM6tI0vzPXi
9AA44Gvy4e/
bYDAfd08oDYkQgtnHjobBVq9MPwRuJyOfNdVjTHagvEFXd5b2Q2Cz3hmpaY0Bw7sPCYOjEX1akUm0N3rg9+Ypeh
LfRS8rC7H26aXxqT9OGOU9FMrbs2ANDvRCiXAV8CA8KkeY2okWiADAjhTzooiwR6aYlk2GuhwwMLPetBjlKUacC
1YchEdlOsCrjtzSRs1XQScAbJ+B4j56wNDVDfmKqsak1i5O2vj7CadrFcA503yAusBIxMSu4wlbqYx9NRtw91h+
gbOGX955Z4mRRfF/15bBw2K59Dmccq1Ogs8R1/tSii9YzXIuvvgGKzNWtkyQ9IvG4xBoeFaUm1WBrklT4gyh/
MtIV9VY9Qfi+nI7fBQo0biJGgqzr4/
oeve+XFMhHjQpfGzmfWW5QldsMGoFjIEdGNq19MqTMmfQUqy7DxtxTA32eqVOn36u5iQWE0pdaAxCnJpJVHd3df
rtoY6OhWVb2oe7erKKqU5nbpEwXopXufkXW+e+g/JOVzWz9pBeyeG7mFmOJVZd",
        "expireInSeconds": 31536000,
        "userId": 10781,
        "user": {
            "name": "Elshan",
            "surname": "Rasul",
            "middleName": null,
            "companyName": null,
            "userName": "ElshanRasul",
            "emailAddress": "elshan@yoll.io",
            "phoneNumber": "9999999999",
            "address": "22222 ABC road",
            "address2": null,
```

```
            "countryState": "North Carolina",
            "city": "Raleigh",
            "postalCode": "27606",
            "tenantName": "",
            "accountType": 30,
            "birthDate": "01/01/1990",
            "businessType": "Financial",
            "profileImageUrl": null,
            "numberOfEmployees": null,
            "registrationNumber": "29892F7",
            "approximateAGI": null,
            "businessTypeId": 3,
            "countryStateId": 33
        }
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{
    "result": {
        "otherExpenses": 9714.72,
        "mealExpenses": 0.0,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 0.0,
        "travelExpenses": 4018.80,
        "giftExpenses": 482.85,
        "deductibleGiftExpenses": 38.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
true
true
false
{otherExpenses=9714.72, mealExpenses=0.0, entertainmentExpenses=0.0,
mealAndEntertainmentExpenses=0.0, travelExpenses=4018.8, giftExpenses=482.85,
deductibleGiftExpenses=38.99, taxSaving=0.0, numberOfEmployees=0}
```


Iterating through keys and printing key-value pairs


```
otherExpenses: 9714.72
mealExpenses: 0.0
entertainmentExpenses: 0.0
mealAndEntertainmentExpenses: 0.0
travelExpenses: 4018.8
giftExpenses: 482.85
deductibleGiftExpenses: 38.99
taxSaving: 0.0
numberOfEmployees: 0
```

Printing all keys

```
[otherExpenses, mealExpenses, entertainmentExpenses, mealAndEntertainmentExpenses, travelExpenses, giftExpenses, deductibleGiftExpenses, taxSaving, numberOfEmployees]
```

Printing all values

```
[9714.72, 0.0, 0.0, 0.0, 4018.8, 482.85, 38.99, 0.0, 0]
```

Set in Java

A **Set** in Java is a **collection** that does **not allow duplicate elements**. It is part of the **Java Collections Framework** and is implemented by various classes such as HashSet, LinkedHashSet, and TreeSet.

 Key Features of a Set

1. **No Duplicates** – A Set does **not allow duplicate elements**.
2. **Unordered (in HashSet)** – Elements are **not stored in any particular order** (except for TreeSet and LinkedHashSet).
3. **Allows Null (in HashSet)** – HashSet allows **one** null value.
4. **Efficient Lookup** – Searching for an element in a HashSet is **very fast** (O(1) time complexity).
5. **Implements Collection Interface** – Set extends Collection, meaning it has common methods like add(), remove(), contains(), etc.
6. **Extracted the keys using keySet()** and stored them in a Set<String> keySet.
7. **Iterated through the keys**:
8. Retrieved and printed the corresponding value using dashboardData.get(key).
9. **Printed all keys** using dashboardData.keySet().
10. **Printed all values** using dashboardData.values().

Example: Using Set and keySet() in Java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.Collection;

public class KeySetExample {
    public static void main(String[] args) {
        // Creating a HashMap with key-value pairs
        Map<String, Object> dashboardData = new HashMap<>();
        dashboardData.put("otherExpenses", 1200.50);
        dashboardData.put("mealExpenses", 850.75);
        dashboardData.put("travelExpenses", 2200.30);
```

```
        dashboardData.put("giftExpenses", 430.00);
        dashboardData.put("taxSaving", 500.00);

        // Retrieving all keys using keySet()
        Set<String> keySet = dashboardData.keySet();

        // Printing each key and its corresponding value
        for (String key : keySet) {
            System.out.println(key + ": " + dashboardData.get(key));
        }

        // Printing the complete key set
        System.out.println("All Keys: " + dashboardData.keySet());

        // Printing the values of the map
        System.out.println("All Values: " + dashboardData.values());
    }
}
```

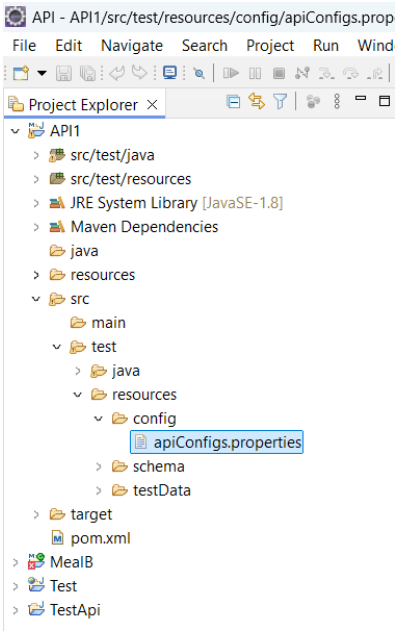
Output:

```
otherExpenses: 1200.5
mealExpenses: 850.75
travelExpenses: 2200.3
giftExpenses: 430.0
taxSaving: 500.0
All Keys: [otherExpenses, mealExpenses, travelExpenses, giftExpenses,
taxSaving]
All Values: [1200.5, 850.75, 2200.3, 430.0, 500.0]
```

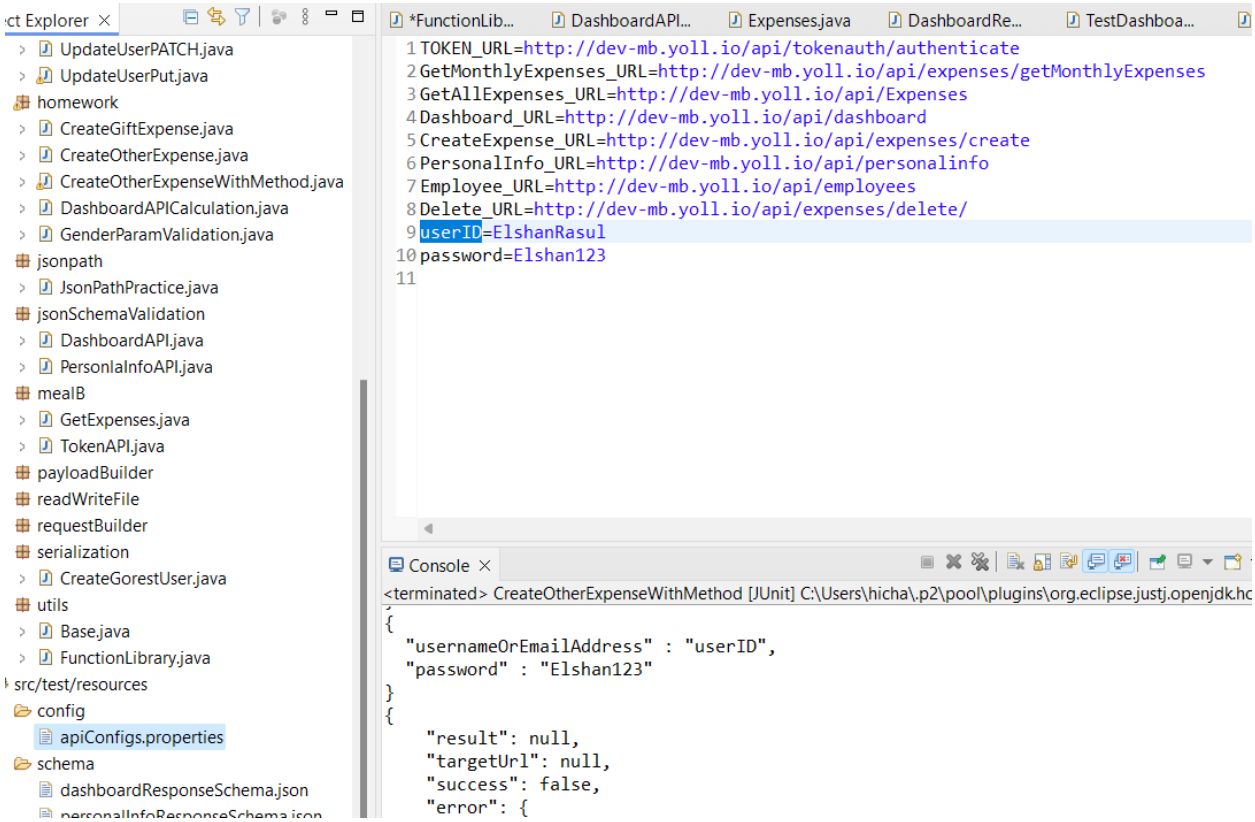
Using apiConfigs.properties file:

This file contains key-value pairs where each key represents an API endpoint or authentication detail, and its corresponding value is the actual URL or credential.

Create a file:`apiConfigs.properties` under the folder (config) containing the urls, userID, and password:




```
TOKEN_URL=http://dev-mb.yoll.io/api/tokenauth/authenticate
GetMonthlyExpenses_URL=http://dev-mb.yoll.io/api/expenses/
getMonthlyExpenses
GetAllExpenses_URL=http://dev-mb.yoll.io/api/Expenses
Dashboard_URL=http://dev-mb.yoll.io/api/dashboard
CreateExpense_URL=http://dev-mb.yoll.io/api/expenses/create
PersonalInfo_URL=http://dev-mb.yoll.io/api/personalinfo
Employee_URL=http://dev-mb.yoll.io/api/employees
Delete_URL=http://dev-mb.yoll.io/api/expenses/delete/
userID=ElshanRasul
password=Elshan123
```



Java Code to Read Properties File: apiConfigs.properties

- To load this properties file in Java, Create a class named Base.

Now, you can easily get the API URLs and credentials from the *apiConfigs.properties* file instead of writing them directly in your Java code!

- Use the following class to facilitate the code by using Variables from *apiConfigs.properties* instead of elements.

- Base class will be the parent of : FunctionLibrary.

```
package utils;

import java.io.File;
import java.io.FileInputStream;
import java.util.Properties;

public class Base {
     public static Properties property;
     static {
          try {
```

```
                property = new Properties();
//put the path :
                String path =
"src/test/resources/config/apiConfigs.properties";
                File file = new File(path);
                FileInputStream fis = new FileInputStream(file);
                property.load(fis);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

Explanation of the `Base` Class Line by Line:

1. Package Declaration

```
package utils;
```

- This line declares that the `Base` class belongs to the `utils` package.

2. Import Statements

```
import java.io.File;
import java.io.FileInputStream;
import java.util.Properties;
```

- These imports are required for handling files and properties:
  - `File`: Represents a file in the system.
  - `FileInputStream`: Reads data from a file.
  - `Properties`: A class for handling key-value pairs (used for reading the `.properties` file).

3. Declaring the `property` Object

```
public static Properties property;
```

- A `Properties` object is declared as `static` so it can be accessed anywhere in the project without needing an instance of the class.

4. Static Block for Initialization

```
static {
    try {
```

- A `static` block is used to load the properties file when the class is first used.
- This ensures the properties are loaded once and can be accessed globally.

5. Defining the Properties File Path

```
String path = "src/test/resources/config/apiConfigs.properties";
```

```
File file = new File(path);
```

- The `path` variable stores the relative path to the `apiConfigs.properties` file.
- `File file = new File(path);` creates a `File` object representing the properties file.

6. Reading the Properties File

```
FileInputStream fis = new FileInputStream(file);
property.load(fis);
```

- `FileInputStream fis = new FileInputStream(file);` opens the file for reading.
- `property.load(fis);` loads the key-value pairs from the file into the `Properties` object.

7. Exception Handling

```
} catch (Exception e) {
    e.printStackTrace();
}
```

- If any issue occurs (e.g., file not found), the error is printed to help with debugging.

How to Use This Class ?

After defining this **Base** class, you can easily access the API URLs and credentials from anywhere in your project, by using:

```
String apiUrl = Base.property.getProperty("Dashboard_URL");
System.out.println(apiUrl);
```

This will print the URL stored in `apiConfigs.properties` for `Dashboard_URL`.

In our case, we use `Base.property.getProperty(" ");` for `CreateExpense_URL` , `TOKEN_URL` , `userID` , `password`.

- We use userID and password instead of ElshanRasul and Elshan12 :

```
//payload.setUsernameOrEmailAddress("ElshanRasul");
//payload.setPassword("Elshan123"); ==➔

payload.setUsernameOrEmailAddress("userID ");

payload.setPassword("password ");
```

and instead of tokenUrl, createExpenseUrl use: TOKEN_URL, CreateExpense_URL

```
//String tokenUrl = "http://dev-mb.yoll.io/api/tokenauth/authenticate";
//String createExpenseUrl="http://dev-mb.yoll.io/api/expenses/create";


/*post(tokenUrl);*/

/*post(createExpenseUrl);*/  ==➔

 .post(property.getProperty("TOKEN_URL"));

.when().post(property.getProperty("CreateExpense_URL"));
```

FunctionLibrary Class using *apiConfigs.properties* file

```
package utils;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.OtherExpensePayloadBuilder;
import payloadBuilder.TokenPayloadBuilder;

public class FunctionLibrary extends Base {
     //String tokenUrl =
"http://dev-mb.yoll.io/api/tokenauth/authenticate";
     String
createExpenseUrl="http://dev-mb.yoll.io/api/expenses/create";


     /**
      * @author  This method submit Get request in MEALB project
      */
     public Response submitGETRequest(String url) {
          Response dashboardResp = given()
```

```
                                                    .header("Authorization",
"Bearer " + getToken())
                                                    .when().get(url);
          dashboardResp.prettyPrint();
          return dashboardResp;
     }

     /**
      * @author ElshanRasulov This method submit POST request to create
expense
      */

     // We must return RESPONSE  object type from Rest Assured, Not
String Method.
//   Response stores the response from the API call, including:
//        Status Code (e.g., 200 OK, 400 Bad Request)
//        Response Body (JSON, XML, etc.)
//        Headers
//        Cookies
//        This allows the calling function to validate or extract data
from the response.
     public Response createExpense(String payload) {
          Response response = given()
                                          .header("Authorization", "Bearer
" + getToken())
                                          .body(payload)
                                          .contentType(ContentType.JSON)
                                     .when().post(property.getProperty("Cr
eateExpense_URL"));
          response.prettyPrint();
          return response;
     }


     /**
      * @author ElshanRasulov This method submits POST request to create
token and
      *         retrieve token from response and return
      */
     public String getToken() {
          Response response =
given().body(createTokenPayload()).contentType(ContentType.JSON).when()
                    /*post(tokenUrl);*/
          .post(property.getProperty("TOKEN_URL"));
          response.prettyPrint();
          assertEquals(200, response.getStatusCode());
//extract "accessToken" from response
          String token = JsonPath.read(response.asString(),
"$.result.accessToken");
          return token;
```

```
	}

	/**
	 * @author ElshanRasulov This method create token request payload
data and
	 *         convert it to string and return
	 */
	public String createTokenPayload() {
		TokenPayloadBuilder payload = new TokenPayloadBuilder();

		payload.setUsernameOrEmailAddress(property.getProperty("userID"));
		payload.setPassword(property.getProperty("password"));
		return convertObjToString(payload);
	}

	/**
	 * @author ElshanRasulov This method convert payload objects to
String format
	 */
	public String convertObjToString(Object obj) {
		String strPayload = null;
		try {
			ObjectMapper objMap = new ObjectMapper();
			strPayload =
objMap.writerWithDefaultPrettyPrinter().writeValueAsString(obj);
			System.out.println(strPayload);
		} catch (JsonProcessingException e) {
			e.printStackTrace();
		}
		return strPayload;
	}

	/**
	 * @author ElshanRasulov This method create Other Expense request
data
	 */
	public String createOtherExpensePayload(String name, String amount,
String dateTime, String id) {
		OtherExpensePayloadBuilder payload = new
OtherExpensePayloadBuilder();
		payload.setName(name);
		payload.setAmount(amount);
		payload.setExpenseDateTime(dateTime);
		payload.setExpenseType("Other");
		payload.setOtherExpenseNameID(id);
		// Use this to return
		// String strPayload=convertObjToString(payload);
		// return strPayload;
		//Or :
		return convertObjToString(payload);
```

```
	}


}
```

GET METHOD:

This method is repeated many times, We use the GET Method in **submitGETRequest()** method, then we can call it

```
public Response submitGETRequest(String url) {
		Response dashboardResp = given()
												.header("Authorization",
"Bearer " + getToken())
												.when().get(url);
		dashboardResp.prettyPrint();
		return dashboardResp;
	}
```

Calling Get Method in TestDashboardResponseMap Class:

```
public class TestDashboardResponseMap extends FunctionLibrary {
	@Test
	public void test() throws JsonMappingException, JsonProcessingException {

		//String dashboardURL = "http://dev-mb.yoll.io/api/dashboard";
		//Response dashboardResp = given().header("Authorization", "Bearer " +
getToken()).when().get(dashboardURL);
		//dashboardResp.prettyPrint();
Response
dashboardResp=submitGETRequest(property.getProperty("Dashboard_URL"));
```

And in public class FunctionLibrary extends Base

```
/**
	 * @author  This method submit Get request in MEALB project

	 */ Return Response Version


	public Response submitGETRequest(String url) {
		Response dashboardResp = given().header("Authorization", "Bearer " +
getToken())
				.when().get(url);
		dashboardResp.prettyPrint();
		return dashboardResp;
	}
```

We can also return String Version. We choose the Response Version.

Response Object:

- `Response` is a class provided by RestAssured that represents the response returned from the server after making an HTTP request.

String Version : no Return

```
public void submitGETRequest(String url) {
    Response dashboardResp=given()
                                .header("Authorization", "Bearer "+getToken())
                            .when()
                                .get(url);
    dashboardResp.prettyPrint();
}
```

Response Version

```
/**
 * @author ElshanRasulov
 * This method submit GET request in MEALB projects
 */
public Response submitGETRequest(String url) {
    Response dashboardResp=given()
                                .header("Authorization", "Bearer "+getToken())
                            .when()
                                .get(url);
    dashboardResp.prettyPrint();
    return dashboardResp;
}
```

We can use The easiest way instead the previous ones:

```
@Test
public void test() throws JsonMappingException, JsonProcessingException {
    String dashboardURL="http://dev-mb.yoll.io/api/dashboard";
    Response dashboardResp=given()
                                .header("Authorization", "Bearer "+getToken())
                            .when()
                                .get(dashboardURL);
    dashboardResp.prettyPrint();
    //Response dashboardResp=submitGETRequest(property.getProperty("Dashboard_URL"));
```

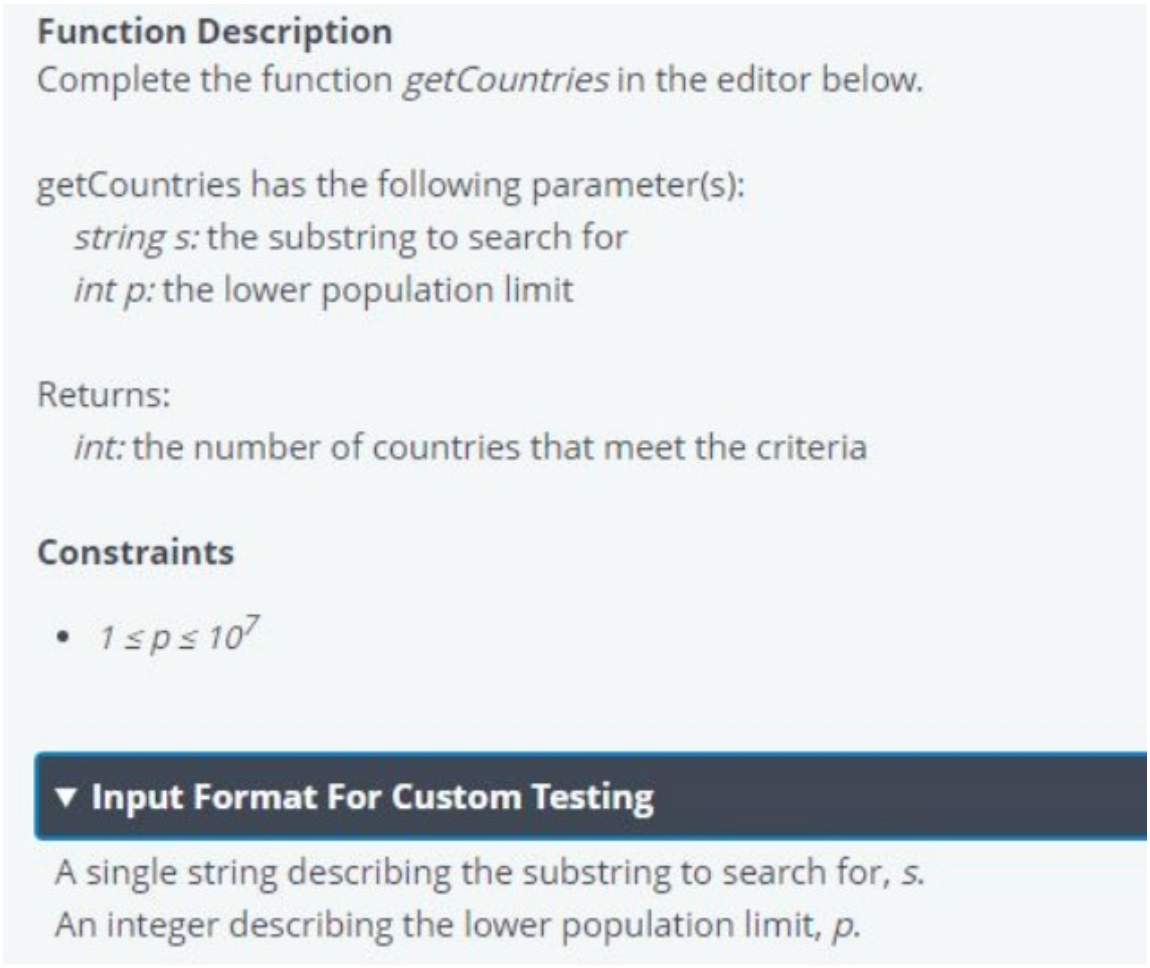
Use the *HTTP GET* method to retrieve information from a database of countries. Query *https://jsonmock.hackerrank.com/api/countries/search?name=s* where *s* is the value of *s*, to find all of the countries that have the substring in their names (not case sensitive). The query response is paginated and can be further accessed by appending to the query string &*page=num* where *num* is the page number.

The query response from the website is a JSON response with the following fields:
page: current page number
per_page: maximum number of records to display per page
total: total records matching the query
total_pages: the number of pages required to display the matching records
data: a json array with the following fields

- *altSpellings*: Alternative spellings of the country's name
- *area*: The country's area (in km^2)
- *capital*: The name of the country's capital
- *currencies*: The country's official currency
- *latlng*: The capital's latitude and longitude
- *name*: The country's name
- *population*: The country's estimated population
- *timezones*: The timezones that the country spans across
- *topLevelDomain*: The country's top-level domain extension
- *callingCodes*: The international phone calling codes of the country
- *region*: The country's region within a continent
- *subRegion* : The country's sub-region within a continent
- *nativeName*: The country's native or ancient name
- *languages*: A list of languages spoken in the country

Given a substring *s*, and the minimum population *p*, find the number of country names that contain *s* and that have populations greater than *p*.

Note: the search is NOT case sensitive


Function Description
Complete the function getCountries in the editor below.
getCountries has the following parameter(s):
string s: the substring to search for
int p: the lower population limit
Returns:
int: the number of countries that meet the criteria
Constraints
• $1 \le p \le 10^7$
▼ Input Format For Custom Testing
A single string describing the substring to search for, s.
An integer describing the lower population limit, p.

You need to retrieve country information using an **HTTP GET request** from the API:
 https://jsonmock.hackerrank.com/api/countries/search?name=s

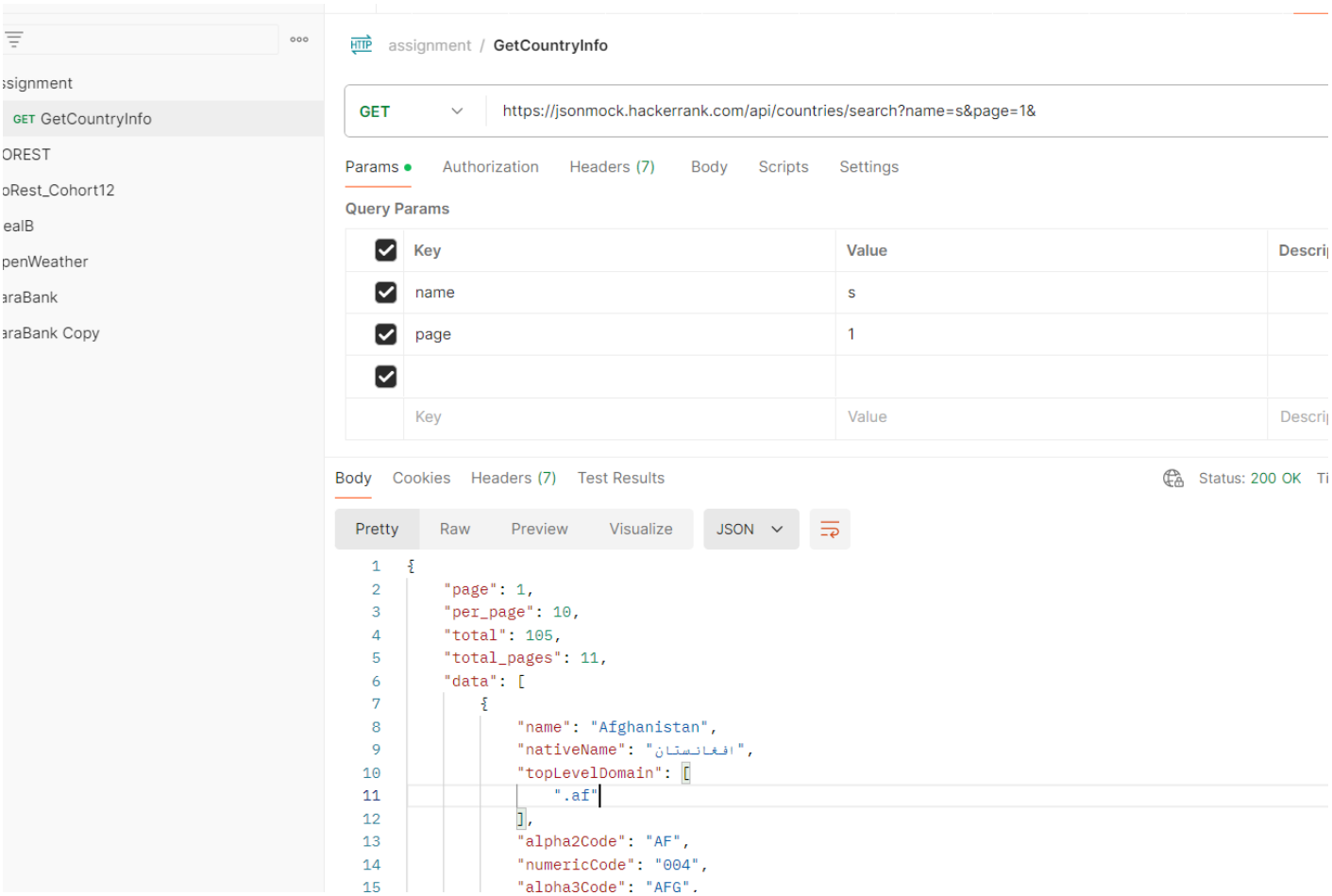
- **Your Task:**

    1. Find **all countries** that contain s in their name.

2. **Filter** them to include only those with a **population greater than p**.
3. Return the **count** of such countries.

Key Points:

- The search is **not case-sensitive**.
- Pagination needs to be handled (&page=num).
- p (population filter) is between **1 and 10 million**.




```
{
    "page": 1,
    "per_page": 10,
    "total": 105,
    "total_pages": 11,
    "data": [
        {
```

```
"name": "Afghanistan",
"nativeName": "افغانستان",
"topLevelDomain": [
    ".af"
],
"alpha2Code": "AF",
"numericCode": "004",
"alpha3Code": "AFG",
"currencies": [
    "AFN"
],
"callingCodes": [
    "93"
],
"capital": "Kabul",
"altSpellings": [
    "AF",
    "Afġānistān"
],
"relevance": "0",
"region": "Asia",
"subregion": "Southern Asia",
"language": [
    "Pashto",
    "Dari"
],
"languages": [
    "ps",
    "uz",
    "tk"
],
"translations": {
    "de": "Afghanistan",
    "es": "Afganistán",
    "fr": "Afghanistan",
    "it": "Afghanistan",
    "ja": "□□□□□□□",
    "nl": "Afghanistan",
    "hr": "Afganistan"
},
"population": 26023100,
"latlng": [
    33,
    65
],
"demonym": "Afghan",
"borders": [
    "IRN",
    "PAK",
    "TKM",
    "UZB",
    "TJK",
```

```
            "CHN"
        ],
        "area": 652230,
        "gini": 27.8,
        "timezones": [
            "UTC+04:30"
        ]
    },
```

```
package homework;

import org.junit.Test;
import com.jayway.jsonpath.JsonPath;
import static io.restassured.RestAssured.given;
import java.util.ArrayList;
import java.util.List;
import io.restassured.response.Response;

public class getCountriesAssignment {
	String url = "https://jsonmock.hackerrank.com/api/countries/search";

	@Test
	public void solutionHW() {

		getCountries("a", 1000000);

	}

	public String getCountryList(String name, int page) {
		Response resp = given().param("name", name).param("page",
page).when().get(url);
		resp.then().statusCode(200);
		return resp.asString();
	}

	public int getCountries(String text, int populationLimit) {
//submit request and get response

//Initialize payload
		String payload = getCountryList(text, 1);
// get count of total pages in response
		int total_pages = JsonPath.read(payload, "$.total_pages");

//Create a list
		List<String> country = new ArrayList<>();
//loop to retrieve countries in all pages
		for (int i = 1; i <= total_pages; i++) {
			payload = getCountryList(text, i);
			List<String> filteredCountries = JsonPath.read(payload,
					"$.data[?(@.population>" + populationLimit +
")].name");
			country.addAll(filteredCountries);
```

```
            }
            System.out.println(country.size());
            System.out.println(country);

            return country.size();
        }
}
```

Result:

137

[Afghanistan, Albania, Algeria, Angola, Argentina, Armenia, Australia, Austria, Azerbaijan, Bahrain, Bangladesh, Belarus, Bolivia, Bosnia and Herzegovina, Botswana, Brazil, Bulgaria, Burkina Faso, Cambodia, Cameroon, Canada, Central African Republic, Chad, China, Colombia, Democratic Republic of the Congo, Costa Rica, Croatia, Cuba, Denmark, Dominican Republic, Ecuador, El Salvador, Equatorial Guinea, Eritrea, Estonia, Ethiopia, Finland, France, Gabon, The Gambia, Georgia, Germany, Ghana, Guatemala, Guinea, Guinea-Bissau, Haiti, Honduras, Hungary, India, Indonesia, Ivory Coast, Iran, Iraq, Republic of Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Liberia, Libya, Lithuania, Republic of Macedonia, Madagascar, Malawi, Malaysia, Mali, Mauritania, Mauritius, Moldova, Mongolia, Mozambique, Myanmar, Namibia, Nepal, Netherlands, New Zealand, Nicaragua, Nigeria, North Korea, Norway, Oman, Pakistan, Palestine, Panama, Papua New Guinea, Paraguay, Poland, Portugal, Qatar, Romania, Russia, Rwanda, Saudi Arabia, Senegal, Serbia, Sierra Leone, Singapore, Slovakia, Slovenia, Somalia, South Africa, South Korea, South Sudan, Spain, Sri Lanka, Sudan, Swaziland, Switzerland, Syria, Taiwan, Tajikistan, Tanzania, Thailand, East Timor, Trinidad and Tobago, Tunisia, Turkmenistan, Uganda, Ukraine, United Arab Emirates, United States, Uruguay, Uzbekistan, Venezuela, Vietnam, Zambia, Zimbabwe]

1. Class Declaration and API Base URL

```
public class getCountriesAssignment {
    String url = "https://jsonmock.hackerrank.com/api/countries/search";
```

- **`public class getCountriesAssignment {`** → Declares the `getCountriesAssignment` class.
- **`String url = "https://jsonmock.hackerrank.com/api/countries/search";`** → Stores the base URL for the API.

---

2. JUnit Test Method

```
@Test
public void solutionHW() {
    getCountries("a", 1000000);
}
```

- **`@Test`** → Marks this method as a JUnit test case.
- **`public void solutionHW() { getCountries("a", 1000000); }`**
    - Calls `getCountries("a", 1000000)`, searching for countries that contain the letter `"a"` and have a population greater than **1,000,000**.

---

3. Method to Retrieve API Response

```
public String getCountryList(String name, int page) {
    Response resp = given()
            .param("name", name)
            .param("page", page)
            .when()
            .get(url);
    resp.then().statusCode(200);
    return resp.asString();
}
```

- **`public String getCountryList(String name, int page)`** → Fetches country data based on a given search term (`name`) and a specific page (`page`).
- **`given().param("name", name).param("page", page)`** → Sends query parameters **"name"** and **"page"** to the API request.
- **`when().get(url)`** → Executes a `GET` request to `url`.
- **`resp.then().statusCode(200);`** → Asserts that the response status is **200 OK**.
- **`return resp.asString();`** → Returns the response as a **String**.

---

4. Method to Get Matching Countries

```
public int getCountries(String text, int populationLimit) {
    // submit request and get response
    String payload = getCountryList(text, 1);
```

- **`String payload = getCountryList(text, 1);`** → Calls `getCountryList()` to get the first page of the API response.

Breaking Down `payload = getCountryList(text, i);`

- **`payload` is a variable that stores the API response.**
- **`text` is the substring to search for in country names.**
- **`i` is the page number (used in a loop to fetch all pages).**

Example Scenario

Let's say:

```
getCountries("a", 1000000);
```

- We search for **countries that contain "a"** and have a **population greater than 1,000,000**.

1st Iteration (`i = 1`):

```
payload = getCountryList("a", 1);
```

- The API fetches **page 1** of results.
- `payload` stores the JSON response **as a String**.

2nd Iteration (`i = 2`):

```
payload = getCountryList("a", 2);
```

- The API fetches **page 2** of results.
- `payload` is updated with the new response.

This continues **until all pages are processed**.

-

---

5. Extract Total Pages

```
// get count of total pages in response
int total_pages = JsonPath.read(payload, "$.total_pages");
```

- **`JsonPath.read(payload, "$.total_pages")`** → Extracts the **total number of pages** from the API response.

---

6. Create a List to Store Country Names

```
// Create a list
List<String> country = new ArrayList<>();
```

- **`List<String> country = new ArrayList<>();`** → Initializes an empty list to store country names.

---

7. Loop Through All Pages to Collect Matching Countries

```
// loop to retrieve countries in all pages
for (int i = 1; i <= total_pages; i++) {
    payload = getCountryList(text, i);
    List<String> filteredCountries = JsonPath.read(payload,
            "$.data[?(@.population>" + populationLimit + ")].name");
    country.addAll(filteredCountries);
}
```

- **`for (int i = 1; i <= total_pages; i++) {`** → Loops through all available pages.
- **`payload = getCountryList(text, i);`** → Retrieves country data for page `i`.

- **`JsonPath.read(payload, "$.data[?(@.population>" + populationLimit + ")].name");`**
  - Extracts **country names** where `population > populationLimit`.
- **`country.addAll(filteredCountries);`** → Adds the filtered country names to the list.

---

8. Print the Number of Matching Countries

```
System.out.println(country.size());
System.out.println(country);
```

- **`System.out.println(country.size());`** → Prints the total count of matching countries.
- **`System.out.println(country);`** → Prints the list of country names.

---

9. Return the Count

```
return country.size();
```

- **Returns** the total number of countries that meet the criteria.

Encoding / Decoding

1 What is Encoding?

- **Encoding** is the process of **converting data** (text, symbols, or numbers) into a different format using a specific scheme.
- It ensures **data compatibility** and safe transmission over different systems.

Example:

- **Text → Binary (UTF-8 encoding)**
  - `"Hello"` → `01001000 01100101 01101100 01101100 01101111`
- **URL Encoding**
  - Special characters are replaced for safe transmission.
  - `"Hello World!"` → `"Hello%20World%21"`

2 What is Decoding?

- **Decoding** is the reverse process of encoding, where **the encoded data is converted back** to its original format.

Example:

- **Binary → Text**
  - `01001000 01100101 01101100 01101100 01101111` → `"Hello"`
- **URL Decoding**
  - `"Hello%20World%21"` → `"Hello World!"`

Common Encoding Types:

1. **Base64 Encoding** → Converts binary data into readable characters (used in authentication).
    - `"Hello"` → `"SGVsbG8="`
2. **URL Encoding** → Converts special characters for safe web transmission.
    - `"space &"` → `"space%20%26"`
3. **UTF-8 Encoding** → Represents characters in binary format (used for text storage).
    - `"A"` → `01000001`

Why is Encoding/Decoding Important?

✔ Secure data transmission
✔ Avoid corruption when transferring data across different systems
✔ Ensure compatibility across different platforms

Example : decoding Password

To encrypte password

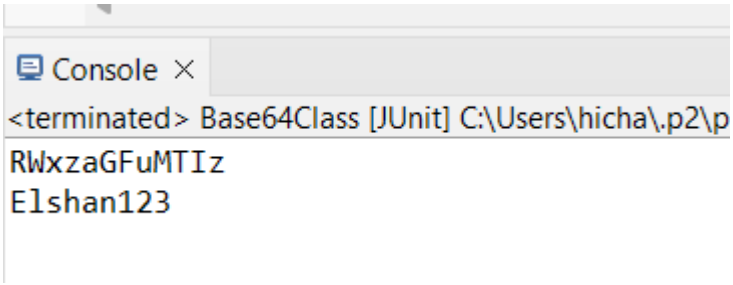
Use this WebSite : Java Base64 Encoding and Decoding | Baeldung

Java supports Base64 encoding and decoding features through the ***java.util.Base64*** utility class.

```
package utils;
//Imports the Base64 class, which provides methods for encoding and decoding.
import java.util.Base64;

import org.junit.Test;

public class Base64Class

//encoding

{
    public String encode(String text) {
        String encodedString = Base64.getEncoder().encodeToString(text.getBytes());
```

```
        return encodedString;
        }
//decoding
        public String decode(String encodedString) {
        byte[] decodedBytes =
Base64.getDecoder().decode(encodedString);
        String decodedString = new String(decodedBytes);
        return  decodedString;
        }

        @Test
        public void test1() {
        String password="Elshan123"; //RWxzaGFuMTIz
        System.out.println(encode(password));

        String encodedString="RWxzaGFuMTIz";//Elshan123


        System.out.println(decode(encodedString));


        }

}
```

Result:



How to use it ?

On apiConfigs.properties file:

Instead of using :password=Elshan123

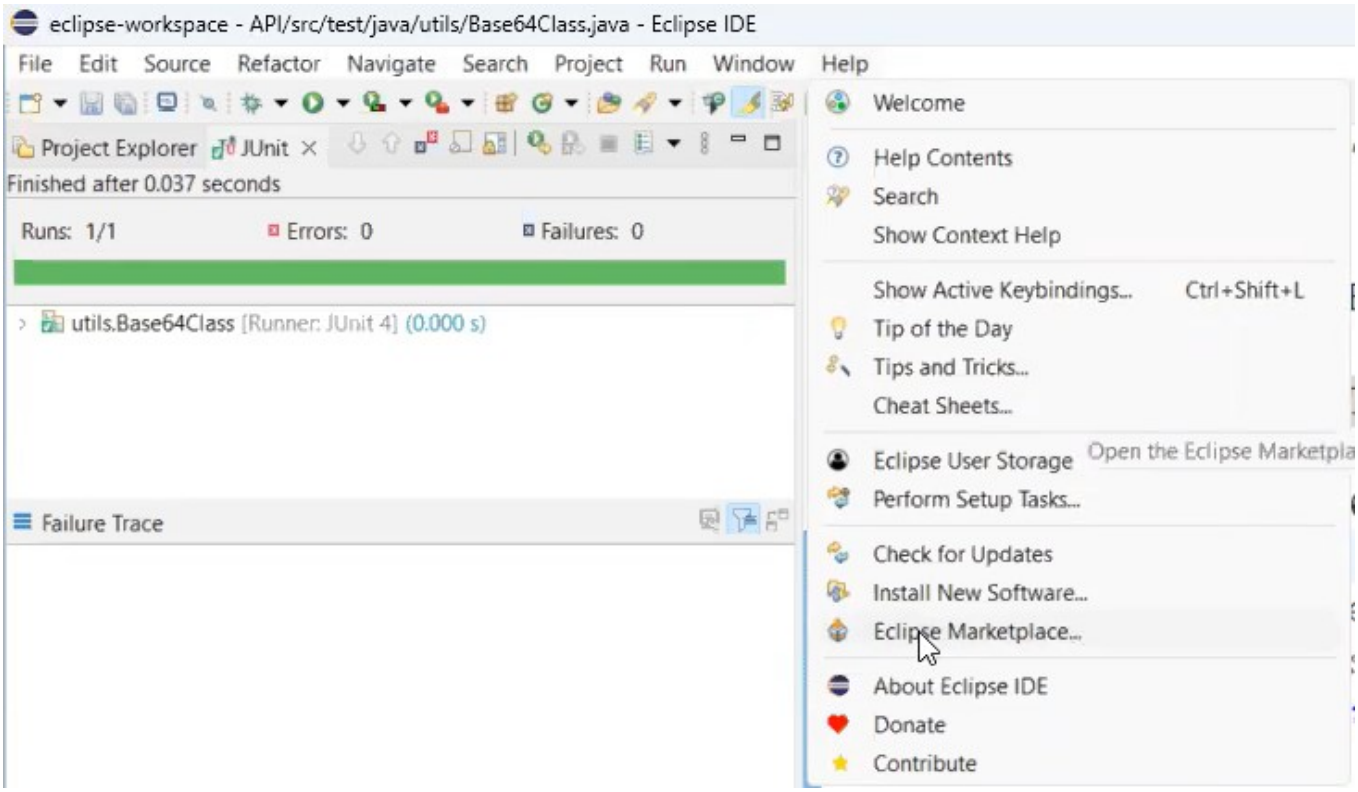
We use: password=RWxzaGFuMTIz

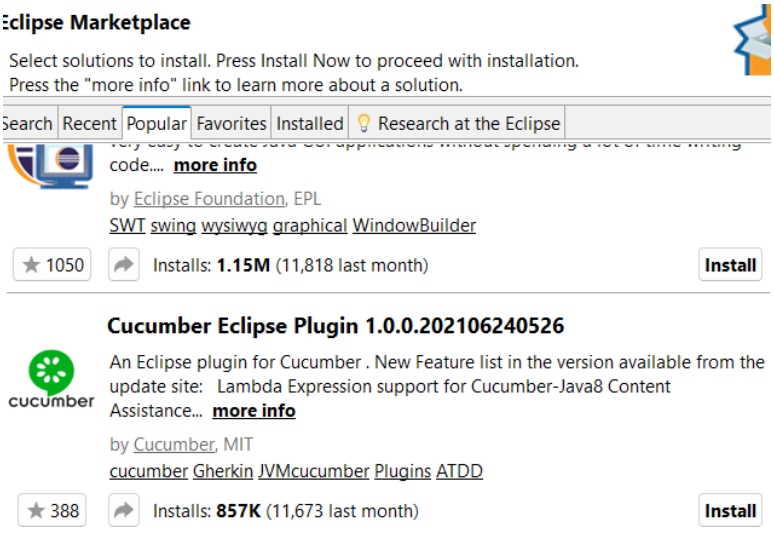
Installing cucumber

Go to **Eclipse/ Help /Eclipse Marketplace /Search** : **Cucumber Eclipse Plugin**

Click on Installed

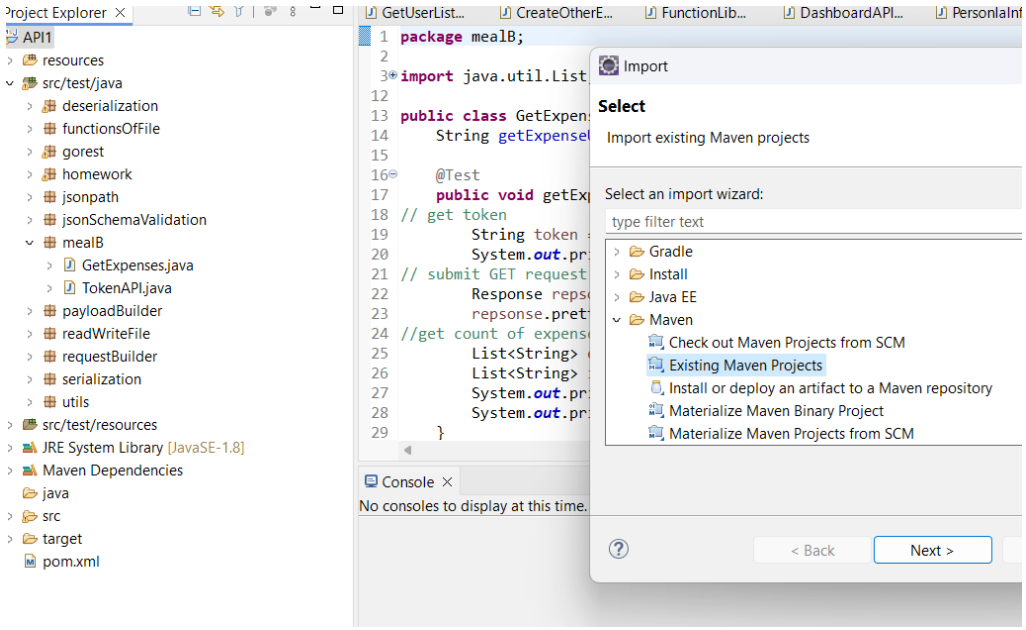
Restart Eclipse



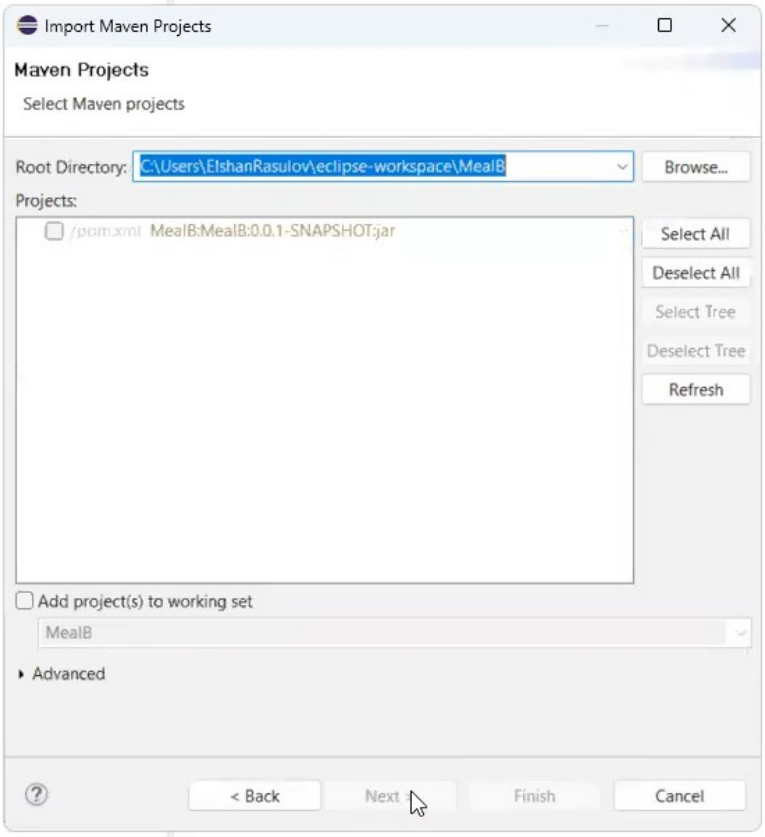

Eclipse Marketplace
Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.
Search Recent Popular Favorites Installed Research at the Eclipse
code.... more info
by Eclipse Foundation, EPL
SWT swing wysiwyg graphical WindowBuilder
★ 1050
Installs: 1.15M (11,818 last month)
Install
Cucumber Eclipse Plugin 1.0.0.202106240526
An Eclipse plugin for Cucumber . New Feature list in the version available from the update site: Lambda Expression support for Cucumber-Java8 Content Assistance... more info
by Cucumber, MIT
cucumber Gherkin JVMcucumber Plugins ATDD
★ 388
Installs: 857K (11,673 last month)
Install

Importing a Maven Project:

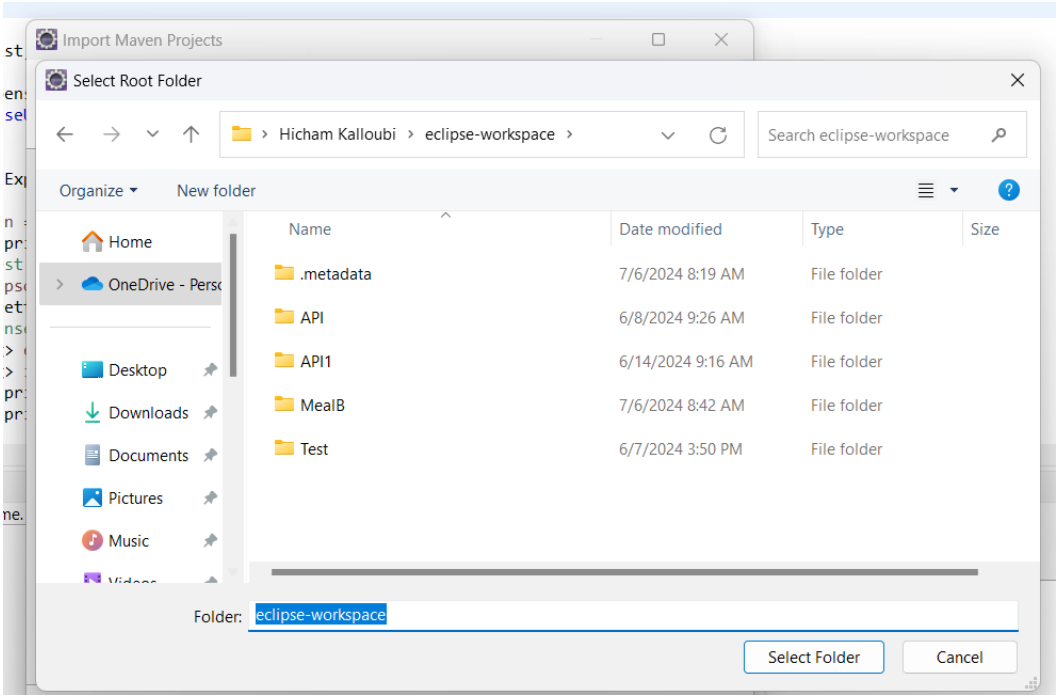
Click on right button, **Import/project/Existing Maven Project / Browse/ select** the folder **from eclipse workspace.**



Select the jar file:

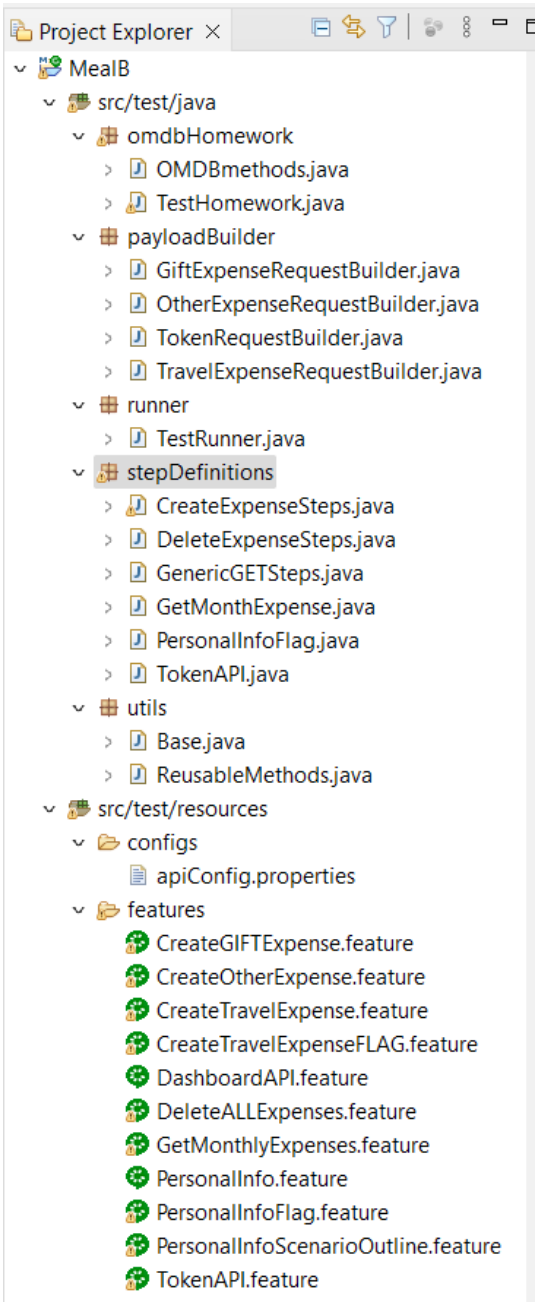


Select a folder



Final Framework: MealB

We will use many Java classes and features files: see the picture



apiConfig.properties file:

```
TOKEN_URL=http://dev-mb.yoll.io/api/tokenauth/authenticate
GetMonthlyExpenses_URL=http://dev-mb.yoll.io/api/expenses/getMonthlyExpenses
GetAllExpenses_URL=http://dev-mb.yoll.io/api/Expenses
Dashboard_URL=http://dev-mb.yoll.io/api/dashboard
CreateExpense_URL=http://dev-mb.yoll.io/api/expenses/create
PersonalInfo_URL=http://dev-mb.yoll.io/api/personalinfo
Employee_URL=http://dev-mb.yoll.io/api/employees
Delete_URL=http://dev-mb.yoll.io/api/expenses/delete/

userID=ElshanRasul
password=Elshan123
password=RWxzaGFuMTIz
```

Base Class:

```
package utils;

import java.io.File;
import java.io.FileInputStream;
import java.util.Properties;

public class Base {
public static Properties property;

      static{
            try {

                  property=new Properties();
                  String path="src/test/resources/configs/apiConfig.properties";
                  File file=new File(path);
                  FileInputStream fis=new FileInputStream(file);
                  property.load(fis);

            }catch(Exception e) {
                  e.printStackTrace();
            }

      }


}
```

ReusableMethods class extended from Base:

```
package utils;

import static io.restassured.RestAssured.given;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
```

```java
import payloadBuilder.GiftExpenseRequestBuilder;
import payloadBuilder.OtherExpenseRequestBuilder;
import payloadBuilder.TokenRequestBuilder;
import payloadBuilder.TravelExpenseRequestBuilder;

public class ReusableMethods extends Base{
	/**
	 * @author ElshanRasulov
	 * @return Response
	 * @see This method submit POST request to delete expenses
	 */
	public Response submitDeleteExpenseRequest(Integer id) {
		Response response=given()
					.header("Authorization", "Bearer "+getToken())
				.when()
					.post(property.getProperty("Delete_URL")+id);
		return response;
	}

	/**
	 * @author ElshanRasulov
	 * @return String
	 * @see This method create payload for TRAVEL type expense and convert it to String
	 */

	public String createTRAVELExpensePayload(String name,String amount, String expenseDateTime){
		TravelExpenseRequestBuilder payload=new TravelExpenseRequestBuilder();
			payload.setName(name);
			payload.setAmount(amount);
			payload.setExpenseDateTime(expenseDateTime);
			payload.setExpenseType("Travel");

		return convertObjectToString(payload);
	}


	/**
	 * @author ElshanRasulov
	 * @return String
	 * @see This method create payload for GIFT type expense and convert it to String
	 */

	public String createGIFTExpensePayload(String name,String amount, String expenseDateTime, String giftRecipient) {

		GiftExpenseRequestBuilder payload=new GiftExpenseRequestBuilder();
		payload.setName(name);
		payload.setAmount(amount);
		payload.setExpenseDateTime(expenseDateTime);
		payload.setExpenseType("Gift");
		payload.setGiftRecipient(giftRecipient);

	return convertObjectToString(payload);

	}
```

```
	/**
	 * @author ElshanRasulov
	 * @return String
	 * @see This method submits POST request to create expense
	 */

	public Response submitCreateExpenseRequest(String payload) {

		Response response=given()
					.header("Authorization", "Bearer
"+getToken())
					.body(payload)
					.contentType(ContentType.JSON)
				.when()
					.post(property.getProperty("CreateExpense_URL
"));
		response.prettyPrint();

		return response;
	}




	/**
	 * @author ElshanRasulov
	 * @return String
	 * @see This method create payload for OTHER type expense and convert it to
String
	 */

	public String createOtherExpensePayload(String name,String amount, String
expenseDateTime,String expenseType, String otherExpenseNameId) {

		OtherExpenseRequestBuilder payload=new OtherExpenseRequestBuilder();
		payload.setName(name);
		payload.setAmount(amount);
		payload.setExpenseType(expenseType);
		payload.setExpenseDateTime(expenseDateTime);
		payload.setOtherExpenseNameId(otherExpenseNameId);

	  return convertObjectToString(payload);

	}



	/**
	 * @author ElshanRasulov
	 * @return String
	 * @see This method submits POST request to generate new token
	 */
	public String getToken() {

		Response tokenResp=given()
					.contentType(ContentType.JSON)
					.body(createTokenPayload())
```

```
                                        .when()
                                            .post(property.getProperty("TOKEN_URL"));
            tokenResp.prettyPrint();

            String token=JsonPath.read(tokenResp.asString(), "$.result.accessToken");

            return token;

      }




      public String createTokenPayload() {

       TokenRequestBuilder tokenReqBuilder=new TokenRequestBuilder();
            tokenReqBuilder.setUsernameOrEmailAddress(property.getProperty("userID"));
            tokenReqBuilder.setPassword(property.getProperty("password"));

            return convertObjectToString(tokenReqBuilder);
      }




      public String convertObjectToString(Object object) {
            String strPayload=null;
            try {
                  ObjectMapper objMapper=new ObjectMapper();
                   strPayload =
objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(object);
            } catch (JsonProcessingException e) {
                  e.printStackTrace();
            }
      return strPayload;
      }


}
```

Explaination

The provided code is a **utility class** named ReusableMethods that extends a Base class. This class contains reusable methods for interacting with REST APIs, creating payloads for different types of requests, and handling JSON serialization. Below is a detailed explanation of the code, broken down into sections:

---

1. Package and Imports

```
package utils;

import static io.restassured.RestAssured.given;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GiftExpenseRequestBuilder;
import payloadBuilder.OtherExpenseRequestBuilder;
import payloadBuilder.TokenRequestBuilder;
import payloadBuilder.TravelExpenseRequestBuilder;
```

- **Package**: The class belongs to the utils package, indicating it contains utility or helper methods.
- **Imports**:
  - **RestAssured**: Used for making HTTP requests and handling responses.
  - **Jackson**: Used for JSON serialization and deserialization.
  - **Jayway JsonPath**: Used for parsing JSON responses and extracting specific data.
  - **Payload Builders**: Custom classes (e.g., GiftExpenseRequestBuilder) used to construct request payloads.

---

2. Class Definition

```
public class ReusableMethods extends Base {
```

- **Class**: ReusableMethods extends a Base class, which likely contains common properties or methods (e.g., property for accessing configuration properties).

---

3. Method: submitDeleteExpenseRequest

```
public Response submitDeleteExpenseRequest(Integer id) {
    Response response = given()
                        .header("Authorization", "Bearer " + getToken())
                    .when()
                        .post(property.getProperty("Delete_URL") + id);
    return response;
}
```

- **Purpose**: Submits a **POST request** to delete an expense by ID.
- **Steps**:
  1. Use **RestAssured** to send a POST request:
     - Add an Authorization header with a Bearer token (retrieved using getToken()).
     - Append the expense ID to the Delete_URL (retrieved from a properties file).
  2. Return the response for further validation or processing.

---

4. Method: createTRAVELExpensePayload

```
public String createTRAVELExpensePayload(String name, String amount, String
expenseDateTime) {
    TravelExpenseRequestBuilder payload = new
TravelExpenseRequestBuilder();
    payload.setName(name);
    payload.setAmount(amount);
    payload.setExpenseDateTime(expenseDateTime);
    payload.setExpenseType("Travel");
    return convertObjectToString(payload);
}
```

- **Purpose**: Creates a JSON payload for a **Travel Expense** request.
- **Steps**:
  1. Create an instance of TravelExpenseRequestBuilder.
  2. Set the required fields (name, amount, expenseDateTime, and expenseType).
  3. Convert the payload object to a JSON string using convertObjectToString.

---

5. Method: createGIFTExpensePayload

```
public String createGIFTExpensePayload(String name, String amount, String
expenseDateTime, String giftRecipient) {
    GiftExpenseRequestBuilder payload = new GiftExpenseRequestBuilder();
    payload.setName(name);
    payload.setAmount(amount);
    payload.setExpenseDateTime(expenseDateTime);
    payload.setExpenseType("Gift");
    payload.setGiftRecipient(giftRecipient);
    return convertObjectToString(payload);
}
```

- **Purpose**: Creates a JSON payload for a **Gift Expense** request.
- **Steps**:
    1. Create an instance of GiftExpenseRequestBuilder.
    2. Set the required fields
       (name, amount, expenseDateTime, expenseType, and giftRecipient).
    3. Convert the payload object to a JSON string
       using convertObjectToString.

---

6. Method: submitCreateExpenseRequest

```
public Response submitCreateExpenseRequest(String payload) {
    Response response = given()
                    .header("Authorization", "Bearer " + getToken())
                    .body(payload)
                    .contentType(ContentType.JSON)
                .when()
                    .post(property.getProperty("CreateExpense_URL"));
    response.prettyPrint();
    return response;
}
```

- **Purpose**: Submits a **POST request** to create an expense.
- **Steps**:
    1. Use **RestAssured** to send a POST request:
        - Add an Authorization header with a Bearer token.
        - Set the request body to the provided JSON payload.
        - Set the content type to application/json.
    2. Print the response in a readable format using response.prettyPrint().
    3. Return the response for further validation or processing.

---

7. Method: createOtherExpensePayload

```
public String createOtherExpensePayload(String name, String amount, String
expenseDateTime, String expenseType, String otherExpenseNameId) {
    OtherExpenseRequestBuilder payload = new OtherExpenseRequestBuilder();
    payload.setName(name);
    payload.setAmount(amount);
    payload.setExpenseType(expenseType);
    payload.setExpenseDateTime(expenseDateTime);
    payload.setOtherExpenseNameId(otherExpenseNameId);
    return convertObjectToString(payload);
}
```

- **Purpose**: Creates a JSON payload for an **Other Expense** request.
- **Steps**:
    1. Create an instance of OtherExpenseRequestBuilder.
    2. Set the required fields (name, amount, expenseDateTime, expenseType, and otherExpenseNameId).
    3. Convert the payload object to a JSON string using convertObjectToString.

---

8. Method: getToken

```
public String getToken() {
    Response tokenResp = given()
                    .contentType(ContentType.JSON)
                    .body(createTokenPayload())
                .when()
                    .post(property.getProperty("TOKEN_URL"));
    tokenResp.prettyPrint();
    String token = JsonPath.read(tokenResp.asString(), "$.result.accessToken");
    return token;
}
```

- **Purpose**: Submits a **POST request** to generate a new access token.
- **Steps**:
    1. Use **RestAssured** to send a POST request:
        - Set the content type to application/json.
        - Set the request body to the token payload (created using createTokenPayload).
    2. Print the response in a readable format using response.prettyPrint().
    3. Extract the access token from the response using **JsonPath** ($.result.accessToken).
    4. Return the access token.

---

9. Method: createTokenPayload

```
public String createTokenPayload() {
    TokenRequestBuilder tokenReqBuilder = new TokenRequestBuilder();

tokenReqBuilder.setUsernameOrEmailAddress(property.getProperty("userID"));
    tokenReqBuilder.setPassword(property.getProperty("password"));
    return convertObjectToString(tokenReqBuilder);
}
```

- **Purpose**: Creates a JSON payload for the token request.
- **Steps**:
    1. Create an instance of TokenRequestBuilder.
    2. Set the required fields (usernameOrEmailAddress and password) using values from the properties file.
    3. Convert the payload object to a JSON string using convertObjectToString.

---

10. Method: convertObjectToString

```
public String convertObjectToString(Object object) {
    String strPayload = null;
    try {
        ObjectMapper objMapper = new ObjectMapper();
        strPayload =
objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(object);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
    return strPayload;
}
```

- **Purpose**: Converts a Java object to a JSON string using **Jackson**.
- **Steps**:
    1. Create an instance of ObjectMapper.
    2. Use writerWithDefaultPrettyPrinter() to format the JSON string.
    3. Handle JsonProcessingException if the conversion fails.

Key Features of the Class

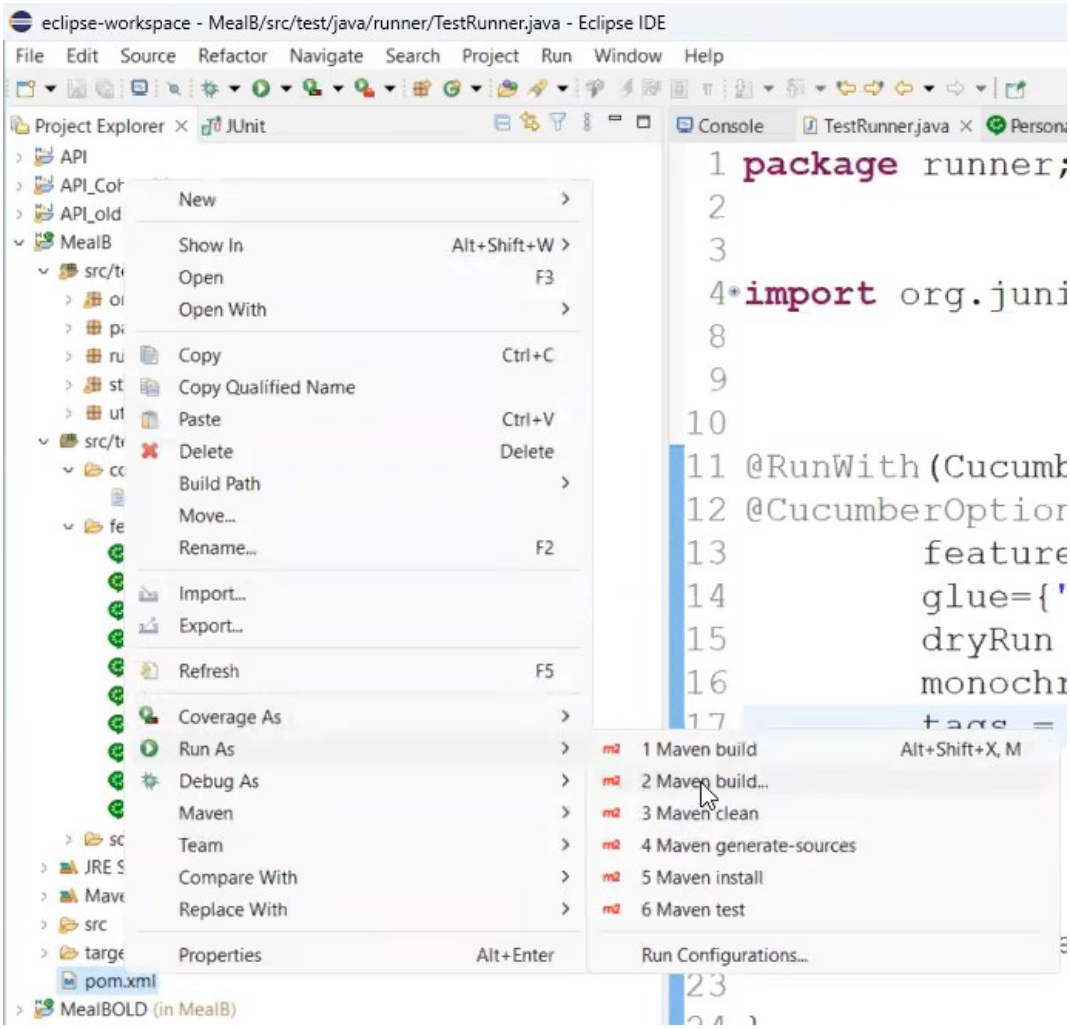
1. **Reusability**:
    - The class provides reusable methods for common tasks like creating payloads, submitting requests, and handling responses.
2. **Modularity**:
    - Each method has a single responsibility, making the code easy to maintain and extend.
3. **JSON Handling**:

- The class uses **Jackson** for JSON serialization and **JsonPath** for parsing JSON responses.

4. **RestAssured Integration**:
   - The class leverages **RestAssured** for making HTTP requests and validating responses.
5. **Configuration Management**:
   - The class uses a properties file (accessed via property.getProperty()) to manage configuration values like URLs and credentials.

How to get the report ?

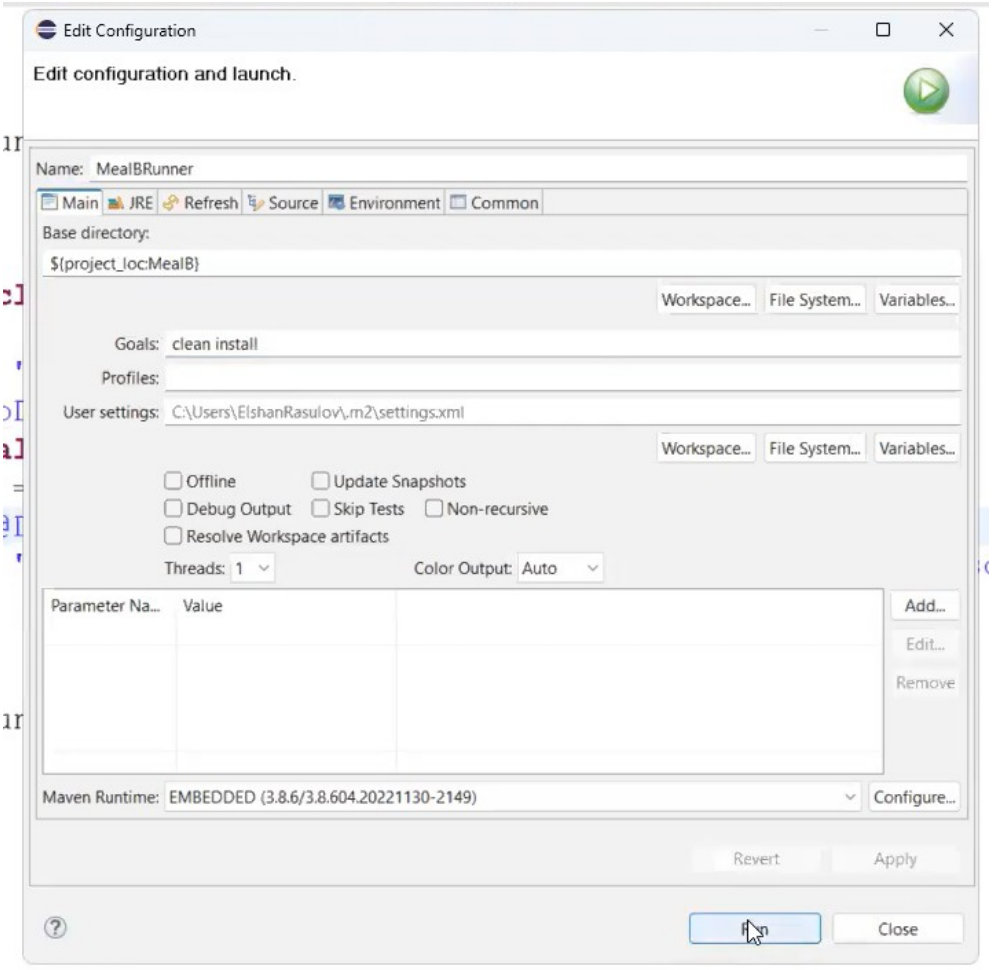
Click right button on pom.xml / Run As / Maven Build



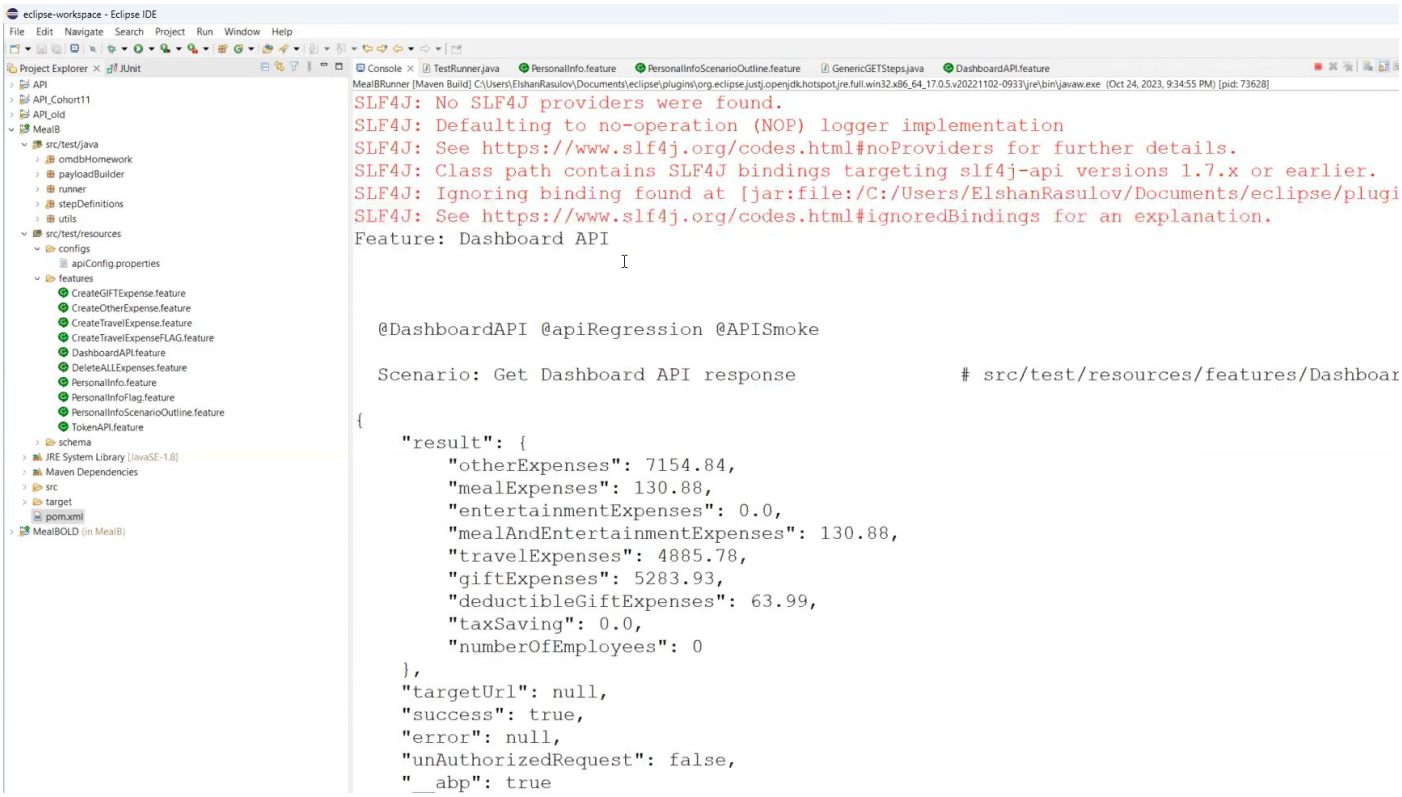
Name: MealBRunner

Goals: clean install

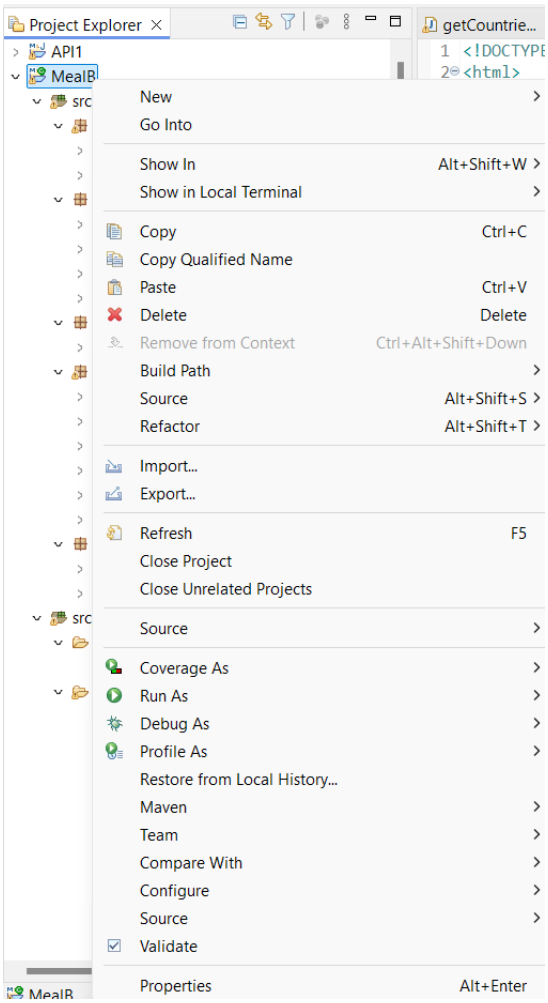
Apply / Run



You will see a log on red and json while creating the report:

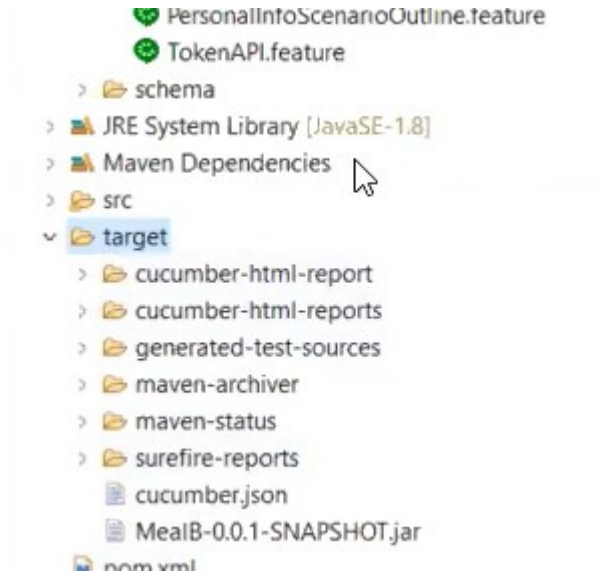


Refresh the target folder by clicking on F5, or :

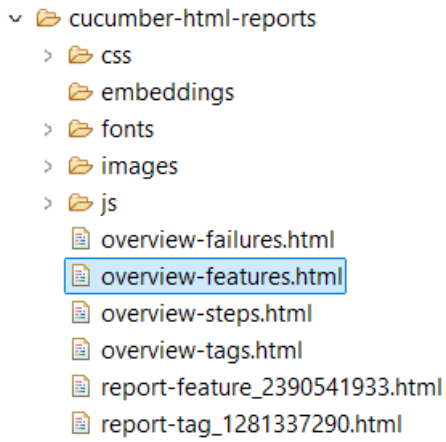


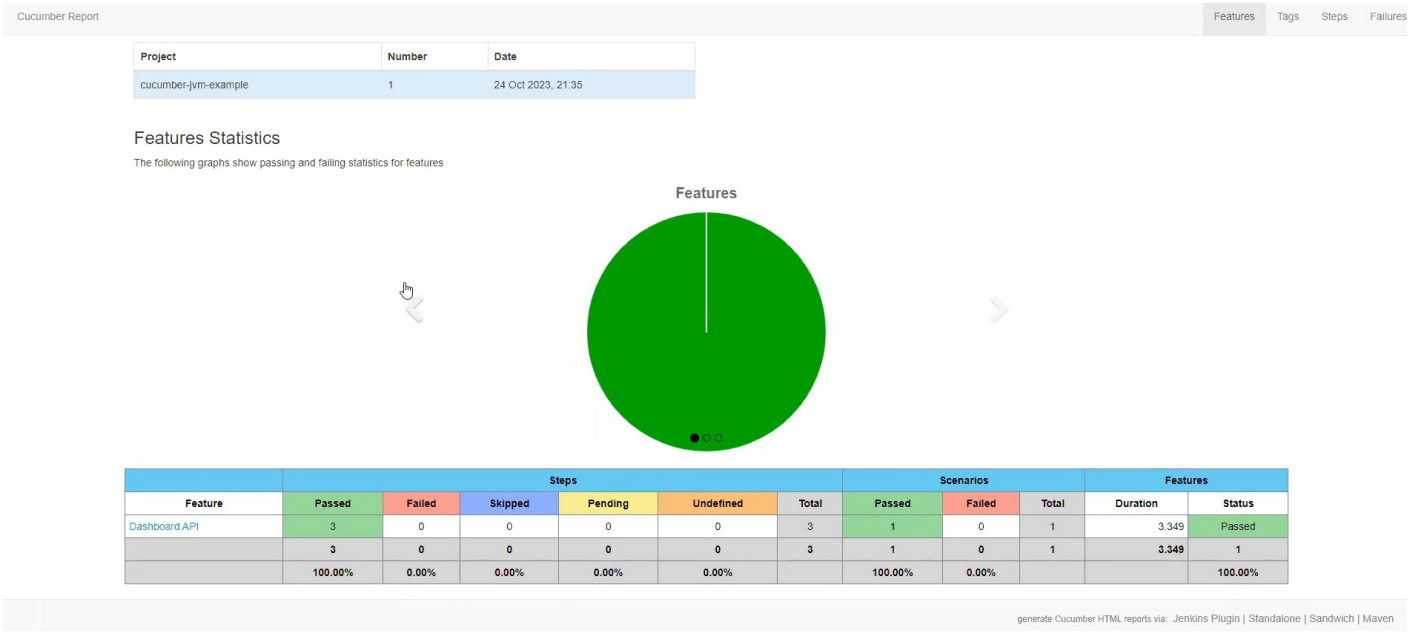
Under target, you will see some folders, select the second folder:

Cucumber-html-reports



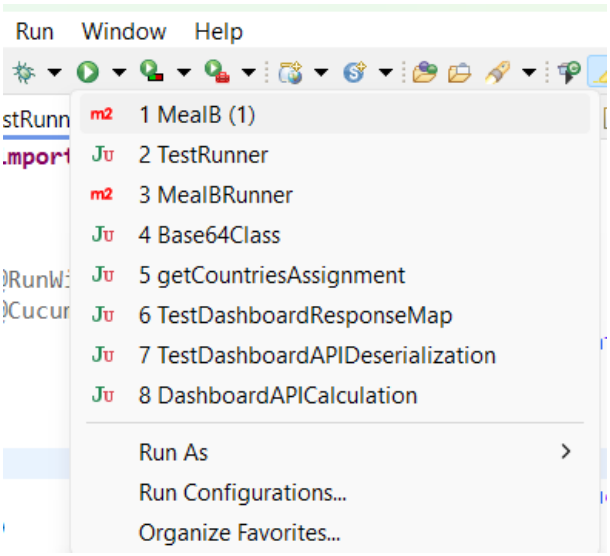
Open overview-features.html



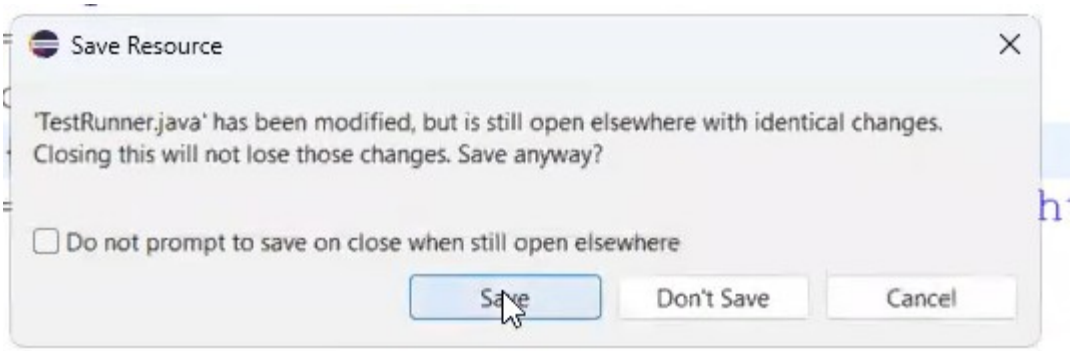


The test must be run with maven, not Junit.

M2 on red color is naven, and Ju is Junit.



Save it.


Save Resource
'TestRunner.java' has been modified, but is still open elsewhere with identical changes.
Closing this will not lose those changes. Save anyway?
Do not prompt to save on close when still open elsewhere
Save
Don't Save
Cancel

TestRunner Class:

Add the tag on TestRunner class: tags = { "@deleteExpense" }
to run the features file with maven

Change the tags for every features file

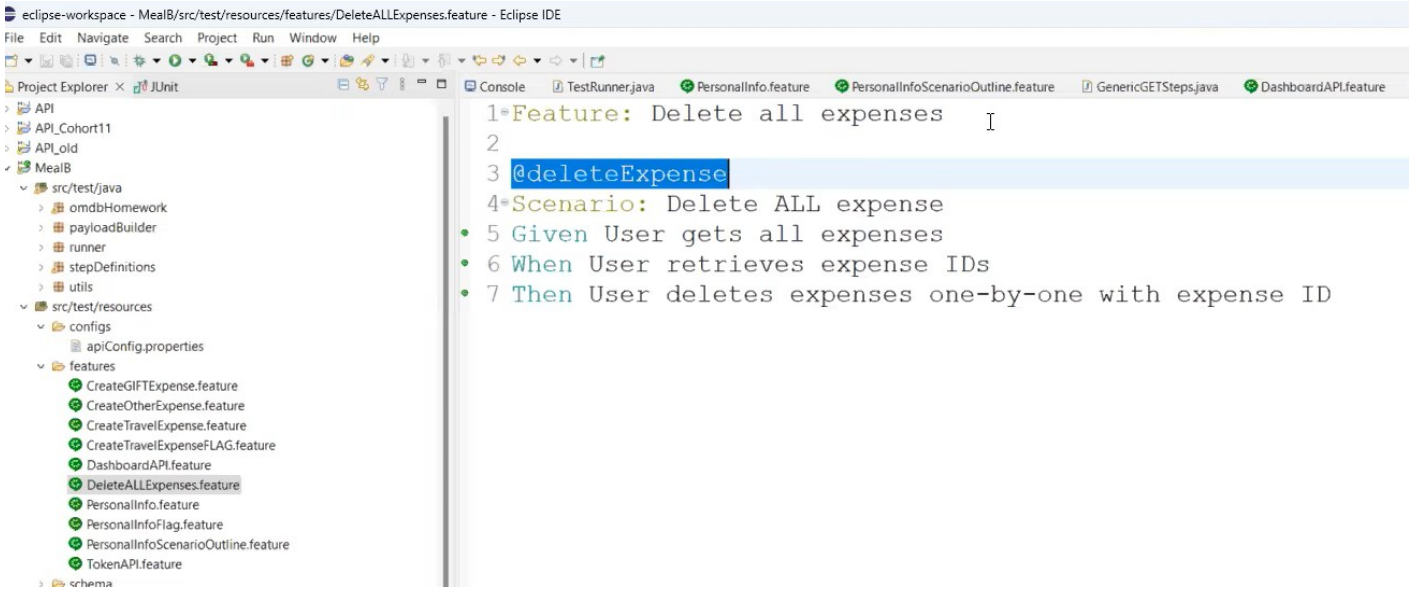
```
package runner;


import org.junit.runner.RunWith;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;



@RunWith(Cucumber.class)
```

```
@CucumberOptions(
        features = "./src/test/resources/features",
        glue={"stepDefinitions"},
        dryRun = false,
        monochrome = true,
        tags = { "@deleteExpense" },
        plugin = { "pretty", "html:target/cucumber-html-report",
"json:target/cucumber.json" }
)


public class TestRunner {

}
```

DeleteALLExpenses.features file:

Put the tag :@deleteExpense on the TestRunner Class

```
Feature: Delete all expenses

@deleteExpense
Scenario: Delete ALL expense
Given User gets all expenses
When User retrieves expense IDs
Then User deletes expenses one-by-one with expense ID
```



DeleteExpenseSteps class extended from ReusableMethods:

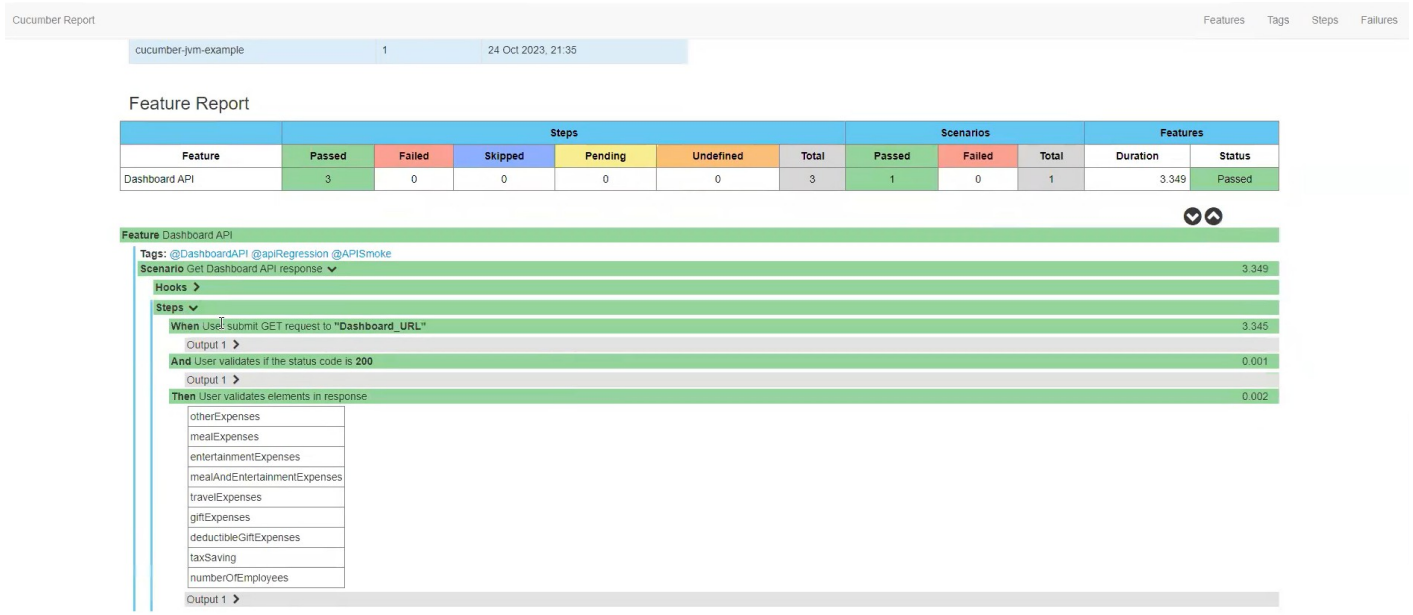
```
package stepDefinitions;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import java.util.List;

import com.jayway.jsonpath.JsonPath;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import io.restassured.response.Response;
import utils.ReusableMethods;

public class DeleteExpenseSteps extends ReusableMethods{
	List<Integer>expenseIDList;
	String payload;
	Response response;
	Scenario scenario;


	@Before
	public void keepScenario(Scenario scenario){
		this.scenario=scenario;
	}

	@Given("User gets all expenses")
	public void user_gets_all_expenses() {
		// Submit GET request
		 response=given()
					.header("Authorization", "Bearer
"+getToken())
				.when()
					.get(property.getProperty("GetAllExpenses_
URL"));
		response.prettyPrint();
		//write value to cucumber report
		scenario.write(response.asPrettyString());
	}

	@When("User retrieves expense IDs")
	public void user_retrieves_expense_IDs() {

		expenseIDList=JsonPath.read(response.asString(),
"$.result[*].id");
		System.out.println(expenseIDList);
	}
```

```
	@Then("User deletes expenses one-by-one with expense ID")
	public void user_deletes_expenses_one_by_one_with_expense_ID() {

		for (Integer eachID : expenseIDList) {
			response=submitDeleteExpenseRequest(eachID);
			assertEquals(200, response.getStatusCode());
			scenario.write("Expense with ID number "+eachID+"
deleted");
		}


	}

}
```

DashboardAPI.features file:

```
Feature: Dashboard API

@DashboardAPI    @apiRegression  @APISmoke
Scenario: Get Dashboard API response

When User submit GET request to "Dashboard_URL"
And User validates if the status code is 200
Then User validates elements in response
		|otherExpenses                |
		|mealExpenses                 |
		|entertainmentExpenses        |
		|mealAndEntertainmentExpenses    |
		|travelExpenses                  |
		|giftExpenses                 |
		|deductibleGiftExpenses       |
		|taxSaving                    |
		|numberOfEmployees            |
```

REPORT:WE CAN SEE THE 3 STEPS (When, and, Then) on the features file and the report



The response is written on the report:

Steps

When User submit GET request to **"Dashboard_URL"**

Output 1


```
{
    "result": {
        "otherExpenses": 7154.84,
        "mealExpenses": 130.88,
        "entertainmentExpenses": 0.0,
        "mealAndEntertainmentExpenses": 130.88,
        "travelExpenses": 4885.78,
        "giftExpenses": 5283.93,
        "deductibleGiftExpenses": 63.99,
        "taxSaving": 0.0,
        "numberOfEmployees": 0
    },
    "targetUrl": null,
    "success": true,
    "error": null,
    "unAuthorizedRequest": false,
    "__abp": true
}
```


And User validates if the status code is **200**

Output 1

```
Expected Status Code: 200---  Actual Status Code: 200
```

Then User validates elements in response

otherExpenses
mealExpenses
entertainmentExpenses
mealAndEntertainmentExpenses
travelExpenses
giftExpenses
deductibleGiftExpenses
taxSaving
numberOfEmployees

Output 1

```
otherExpenses ---- element validated
mealExpenses   ---- element validated
entertainmentExpenses   ---- element validated
mealAndEntertainmentExpenses   ---- element validated
travelExpenses   ---- element validated
giftExpenses   ---- element validated
deductibleGiftExpenses   ---- element validated
taxSaving   ---- element validated
numberOfEmployees   ---- element validated
```

PersonalInfoAPI.features file:
Put the tag :@ PersonalInfo on the TestRunner Class

```
Feature: Personal Info API

@PersonalInfo    @apiRegression  @APISmoke
Scenario: Get Personal Info

When User submit GET request to "PersonalInfo_URL"
And User validates if the status code is 200
Then User validates elements in response
           |name                |
           |surname         |
           |middleName          |
           |companyName         |
           |userName            |
           |emailAddress        |
           |phoneNumber         |
           |address         |
           |address2            |
           |countryState        |
           |city                |
           |postalCode          |
           |tenantName          |
           |accountType         |
           |birthDate           |
           |businessType        |
           |profileImageUrl     |
           |numberOfEmployees   |
           |registrationNumber  |
           |approximateAGI      |
           |businessTypeId      |
           |countryStateId      |
```

GenericGETSteps Class is for:
DashboardAPI.features and PersonalInfoAPI.features files.

```
package stepDefinitions;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.util.List;
import java.util.Map;
import java.util.Set;
```

```java
import com.jayway.jsonpath.JsonPath;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.response.Response;
import utils.ReusableMethods;

public class GenericGETSteps extends ReusableMethods{
static Response response;
     Scenario scenario;

     @Before
     public void keepScenario(Scenario scenario){
          this.scenario=scenario;
     }

     @When("User submit GET request to {string}")
     public void user_submit_GET_request_to(String url) {
                  // Submit GET request
                  response=given()
                                       .header("Authorization", "Bearer
"+getToken())
                                  .when()
                                       .get(property.getProperty(url));
                response.prettyPrint();
                //write value to cucumber report
                scenario.write(response.asPrettyString());

     }


     @When("User validates if the status code is {int}")
     public void user_validates_if_the_status_code_is(int
expectedStatusCode) {
          int actualStatusCode=response.getStatusCode();
          scenario.write("Expected Status Code:
"+expectedStatusCode+"---  Actual Status Code: "+actualStatusCode);
          assertEquals(expectedStatusCode, actualStatusCode);


     }

     @Then("User validates elements in response")
     public void
user_validates_elements_in_response(List<String>elements) {
         // get all elements under "$.result" from response and assign
them to MAP
```

```
        Map<String,Double>
respElements=JsonPath.read(response.asString(), "$.result");
         // from Map get the elements name
        Set<String>keys=respElements.keySet();

        for (String eachElement : elements) {
           assertTrue(keys.contains(eachElement));
           scenario.write(eachElement +"   ---- element validated");
        }



     }

}
```

Explaination of GenericGETSteps Class :

The provided code is a **Cucumber Step Definition** class written in Java for testing REST APIs using **RestAssured**. It defines the steps for submitting a **GET request**, validating the **status code**, and verifying the presence of specific **elements in the response**. Below is a detailed explanation of the code, broken down into sections:

---

1. Package and Imports

```
package stepDefinitions;

import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.util.List;
import java.util.Map;
import java.util.Set;

import com.jayway.jsonpath.JsonPath;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.response.Response;
import utils.ReusableMethods;
```

- **Package**: The class belongs to the stepDefinitions package.
- **Imports**:
  - **RestAssured**: Used for making HTTP requests and validating responses.
  - **JUnit Assertions**: Used for assertions like assertEquals and assertTrue.
  - **Java Collections**: Used for handling lists, maps, and sets.
  - **Jayway JsonPath**: Used for parsing and extracting data from JSON responses.
  - **Cucumber Annotations**: Used for defining step definitions and hooks.
  - **ReusableMethods**: A custom utility class (likely containing reusable methods like getToken()).

---

2. Class Definition and Variables

```
public class GenericGETSteps extends ReusableMethods {
    static Response response;
    Scenario scenario;

    @Before
    public void keepScenario(Scenario scenario) {
        this.scenario = scenario;
    }
```

- **Class**: GenericGETSteps extends ReusableMethods, indicating that it inherits reusable methods from the parent class.
- **Variables**:
  - response: A static Response object to store the API response.
  - scenario: A Scenario object to write logs or data to the Cucumber report.
- **@Before Hook**:
  - The keepScenario method captures the current Cucumber scenario and assigns it to the scenario variable. This allows writing data (e.g., response details) to the Cucumber report.

---

3. Step Definition: Submit GET Request

```
@When("User submit GET request to {string}")
public void user_submit_GET_request_to(String url) {
    // Submit GET request
    response = given()
                .header("Authorization", "Bearer " + getToken())
               .when()
                .get(property.getProperty(url));
    response.prettyPrint();
    // Write value to cucumber report
    scenario.write(response.asPrettyString());
}
```

- **Annotation**: @When defines a Cucumber step for submitting a GET request.
- **Parameters**:
  - url: A placeholder for the endpoint URL, passed from the feature file.

- **Steps**:
  1. Use **RestAssured** to send a GET request:
     - Add an Authorization header with a Bearer token (retrieved using getToken()).
     - Use the get() method to send the request to the URL (retrieved from a properties file using property.getProperty(url)).
  2. Print the response in a readable format using response.prettyPrint().
  3. Write the response to the Cucumber report using scenario.write().

---

4. Step Definition: Validate Status Code

```
@When("User validates if the status code is {int}")
public void user_validates_if_the_status_code_is(int expectedStatusCode) {
    int actualStatusCode = response.getStatusCode();
    scenario.write("Expected Status Code: " + expectedStatusCode + " --- Actual Status Code: " + actualStatusCode);
    assertEquals(expectedStatusCode, actualStatusCode);
}
```

- **Annotation**: @When defines a Cucumber step for validating the status code.
- **Parameters**:
  - expectedStatusCode: The expected HTTP status code (e.g., 200, 404), passed from the feature file.
- **Steps**:
  1. Retrieve the actual status code from the response using response.getStatusCode().
  2. Write the expected and actual status codes to the Cucumber report.
  3. Use assertEquals to compare the expected and actual status codes.

---

5. Step Definition: Validate Elements in Response

```
@Then("User validates elements in response")
public void user_validates_elements_in_response(List<String> elements) {
    // Get all elements under "$.result" from response and assign them to MAP
    Map<String, Double> respElements = JsonPath.read(response.asString(), "$.result");
    // From Map, get the element names
    Set<String> keys = respElements.keySet();

    for (String eachElement : elements) {
        assertTrue(keys.contains(eachElement));
        scenario.write(eachElement + " ---- element validated");
    }
}
```

- **Annotation**: @Then defines a Cucumber step for validating elements in the response.
- **Parameters**:
  - elements: A list of element names to validate, passed from the feature file.

- **Steps**:
    1. Use **JsonPath** to extract all elements under the $.result path from the JSON response and store them in a Map<String, Double>.
    2. Retrieve the keys (element names) from the map using keys = respElements.keySet().
    3. Iterate through the list of expected elements (elements) and check if each element exists in the response
       using assertTrue(keys.contains(eachElement)).
    4. Write a log to the Cucumber report for each validated element.

---

6. Explanation of Key Components

RestAssured

- Used for making HTTP requests and validating responses.
- Example:

```
response = given()
            .header("Authorization", "Bearer " + getToken())
           .when()
            .get(property.getProperty(url));
```

JsonPath

- Used for parsing JSON responses and extracting specific data.
- Example:

```
Map<String, Double> respElements = JsonPath.read(response.asString(),
"$.result");
```

Cucumber Annotations

- @When and @Then define the steps in the feature file.
- @Before is a hook that runs before each scenario to capture the scenario object.

Assertions

- assertEquals(expectedStatusCode, actualStatusCode): Validates that the actual status code matches the expected one.
- assertTrue(keys.contains(eachElement)): Validates that the response contains the expected elements.

Cucumber Reporting

- scenario.write() is used to write logs or data to the Cucumber report, making it easier to debug and analyze test results.

Example Feature File

Here’s an example of how the step definitions might be used in a Cucumber feature file:

```
Feature: Validate GET Request

  Scenario: Validate GET request to /api/users
    When User submit GET request to "usersEndpoint"
    When User validates if the status code is 200
    Then User validates elements in response
      | id    |
      | name |
      | email|
```

How It Works

1. The test starts by submitting a GET request to the /api/users endpoint.
2. It validates that the status code is 200.
3. It checks if the response contains the elements id, name, and email.

Conclusion

This step definition class provides a reusable and modular way to test REST APIs using **Cucumber**, **RestAssured**, and **JsonPath**. It demonstrates how to:

- Submit a GET request.
- Validate the status code.
- Verify the presence of specific elements in the response.
- Write logs to the Cucumber report for better traceability.

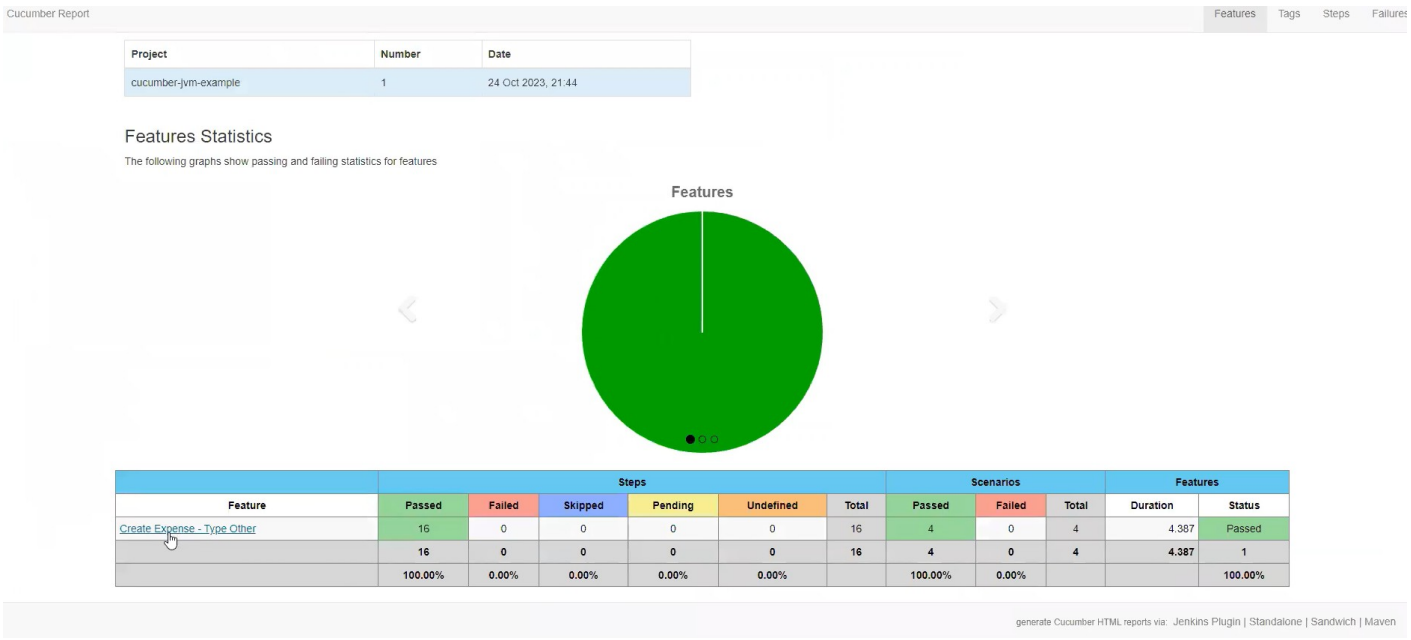
This approach is highly effective for API testing, especially in Behavior-Driven Development (BDD) frameworks like Cucumber.

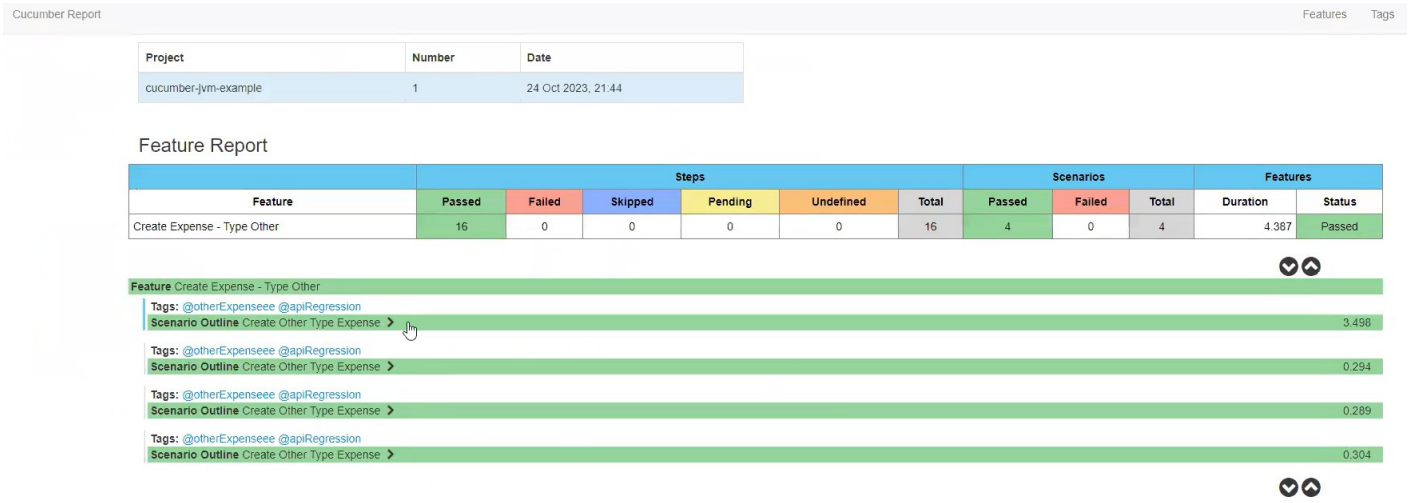
CreateOtherExpense.featues file:

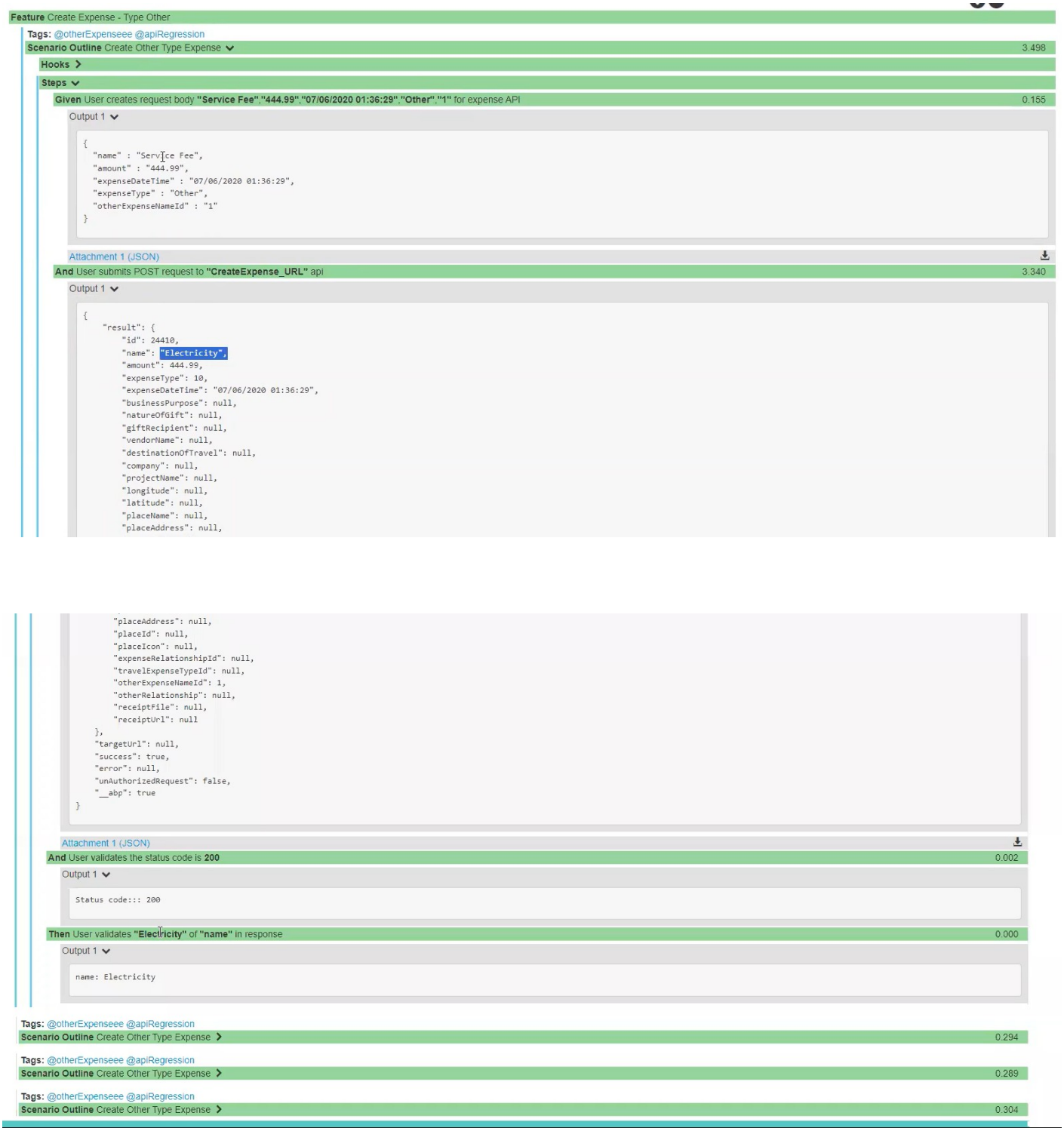
```
Feature: Create Expense - Type Other

  @otherExpenseee @apiRegression
  Scenario Outline: Create Other Type Expense
    Given User creates request body
"<name>","<amount>","<expenseDateTime>","<expenseType>","<otherExpenseNameId>" for
expense API
    And User submits POST request to "CreateExpense_URL" api
    And User validates the status code is 200
    Then User validates "<value>" of "name" in response

    Examples:
      | name         | amount | expenseDateTime     | expenseType | otherExpenseNameId |
value        |
      | Service Fee | 444.99 | 07/06/2020 01:36:29 | Other       |                  1 |
Electricity |
```

```
      | Service Fee | 444.99 | 07/06/2020 01:36:29 | Other       |                  2 |
Rent         |
      | Service Fee | 444.99 | 07/06/2020 01:36:29 | Other       |                  3 |
Gas          |
      | Service Fee | 444.99 | 07/06/2020 01:36:29 | Other       |                  4 |
Service Fee |
```

Report:








```
package stepDefinitions;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GiftExpenseRequestBuilder;
```

```java
import payloadBuilder.OtherExpenseRequestBuilder;
import utils.ReusableMethods;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import com.jayway.jsonpath.JsonPath;

public class CreateExpenseSteps extends ReusableMethods {

	String payload;
	static Response response;
	Scenario scenario;
	String flag="";

	@Before
	public void keepScenario(Scenario scenario){
		this.scenario=scenario;
	}

	@Given("User set {string}")
	public void user_set(String flag) {
	    this.flag=flag;
	}


	@Given("User creates request body {string},{string},{string},{string},{string} for expense API")
	public void user_creates_request_body_for_expense_API(String name, String amount, String expenseDateTime, String expenseType, String otherExpenseNameId) {
		// create payload
		 payload=createOtherExpensePayload(name, amount, expenseDateTime, expenseType, otherExpenseNameId);
		// write request data/payload to report
		 scenario.write(payload);
		 scenario.embed(payload.getBytes(),"application/json" );


	}

	@Given("User submits POST request to {string} api")
	public void user_submits_POST_request_to_api(String url) {

		// Submit POST request to create expense
			response=submitCreateExpenseRequest(payload);
		 // write response to cucumber report
			scenario.write(response.asPrettyString());

scenario.embed(response.asPrettyString().getBytes(),"application/json" );



	}

	@Given("User validates the status code is {int}")
	public void user_validates_the_status_code_is(int expectedStatusCode) {

			int actualStatusCode=response.getStatusCode();
			scenario.write("Status code::: "+actualStatusCode);
```

```
                  assertEquals(expectedStatusCode, actualStatusCode);

      }


      @Then("User validates {string} of {string} in response")
      public void user_validates_of_in_response(String value, String element) {

            String nameValue=JsonPath.read(response.asString(), "$.result.name");
            scenario.write("name: "+nameValue);
            assertEquals(value, nameValue);

      }

      @Given("User creates GIFT request body {string},{string},{string},{string} for
GIFT expense API")
      public void user_creates_GIFT_request_body_for_GIFT_expense_API(String name,
String amount, String expenseDateTime, String giftRecipient) {
            // create payload
              payload=createGIFTExpensePayload(name, amount, expenseDateTime,
giftRecipient);

            // write request data/payload to report
             scenario.write(payload);
             scenario.embed(payload.getBytes(),"application/json" );

      }

      @Given("User creates TRAVEL request body {string},{string},{string} for TRAVEL
expense API")
      public void user_creates_TRAVEL_request_body_for_TRAVEL_expense_API(String name,
String amount, String expenseDateTime) {
            // create payload
              payload=createTRAVELExpensePayload(name, amount, expenseDateTime);

            // write request data/payload to report
             scenario.write(payload);
             scenario.embed(payload.getBytes(),"application/json" );

      }



      @Given("User validate each {string} in response")
      public void user_validate_each_in_response(String element) {
            String jsonpath="$.result."+element;
            Object value=JsonPath.read(response.asString(), jsonpath);
            scenario.write(element+" value: "+value);
      }

}
```

The provided code is a **Cucumber Step Definition** class named CreateExpenseSteps. It defines the steps for creating and validating expenses in an API, specifically for **GIFT**, **TRAVEL**, and **OTHER** expense types. The class extends ReusableMethods, which contains reusable utility methods for API testing. Below is a detailed explanation of the code, broken down into sections:

1. Package and Imports

```
package stepDefinitions;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GiftExpenseRequestBuilder;
import payloadBuilder.OtherExpenseRequestBuilder;
import utils.ReusableMethods;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import com.jayway.jsonpath.JsonPath;
```

- **Package**: The class belongs to the `stepDefinitions` package, which is standard for Cucumber step definitions.
- **Imports**:
  - **Cucumber Annotations**: Used for defining step definitions and hooks.
  - **RestAssured**: Used for making HTTP requests and handling responses.
  - **Payload Builders**: Custom classes (e.g., `GiftExpenseRequestBuilder`) used to construct request payloads.
  - **ReusableMethods**: A utility class containing reusable methods for API testing.
  - **Assertions**: Used for validating responses (e.g., `assertEquals`).
  - **JsonPath**: Used for parsing JSON responses and extracting specific data.

---

2. Class Definition and Variables

```
public class CreateExpenseSteps extends ReusableMethods {
    String payload;
    static Response response;
    Scenario scenario;
    String flag = "";
```

- **Class**: `CreateExpenseSteps` extends `ReusableMethods`, inheriting reusable methods for API testing.
- **Variables**:
  - `payload`: Stores the JSON payload for the API request.
  - `response`: Stores the API response.
  - `scenario`: Used to write logs or data to the Cucumber report.
  - `flag`: A string variable to store a flag or condition (used in the `user_set` method).

---

3. @Before Hook

```
@Before
public void keepScenario(Scenario scenario) {
    this.scenario = scenario;
```

```
}
```

- **Purpose**: Captures the current Cucumber scenario and assigns it to the `scenario` variable.
- **Use Case**: This allows writing logs or embedding data (e.g., request/response) into the Cucumber report.

---

4. Step Definition: Set Flag

```
@Given("User set {string}")
public void user_set(String flag) {
    this.flag = flag;
}
```

- **Annotation**: `@Given` defines a Cucumber step for setting a flag or condition.
- **Parameters**:
  - `flag`: A string value passed from the feature file.
- **Use Case**: This step can be used to set a condition or context for subsequent steps (e.g., enabling/disabling certain validations).

---

5. Step Definition: Create Request Body for OTHER Expense

```
@Given("User creates request body {string},{string},{string},{string},{string} for expense API")
public void user_creates_request_body_for_expense_API(String name, String amount, String
expenseDateTime, String expenseType, String otherExpenseNameId) {
    // Create payload
    payload = createOtherExpensePayload(name, amount, expenseDateTime, expenseType,
otherExpenseNameId);
    // Write request data/payload to report
    scenario.write(payload);
    scenario.embed(payload.getBytes(), "application/json");
}
```

- **Annotation**: `@Given` defines a Cucumber step for creating a request payload for an **OTHER** expense.
- **Parameters**:
  - `name`, `amount`, `expenseDateTime`, `expenseType`, `otherExpenseNameId`: Values passed from the feature file to construct the payload.
- **Steps**:
  1. Call `createOtherExpensePayload` (from `ReusableMethods`) to create the JSON payload.
  2. Write the payload to the Cucumber report using `scenario.write()`.
  3. Embed the payload in the Cucumber report as JSON using `scenario.embed()`.

---

6. Step Definition: Submit POST Request

```
@Given("User submits POST request to {string} api")
public void user_submits_POST_request_to_api(String url) {
    // Submit POST request to create expense
    response = submitCreateExpenseRequest(payload);
    // Write response to cucumber report
    scenario.write(response.asPrettyString());
    scenario.embed(response.asPrettyString().getBytes(), "application/json");
```

```
}
```

- **Annotation**: @Given defines a Cucumber step for submitting a POST request.
- **Parameters**:
  - url: The API endpoint, passed from the feature file.
- **Steps**:
  1. Call submitCreateExpenseRequest (from ReusableMethods) to submit the POST request with the payload.
  2. Write the response to the Cucumber report using scenario.write().
  3. Embed the response in the Cucumber report as JSON using scenario.embed().

---

7. Step Definition: Validate Status Code

```
@Given("User validates the status code is {int}")
public void user_validates_the_status_code_is(int expectedStatusCode) {
    int actualStatusCode = response.getStatusCode();
    scenario.write("Status code::: " + actualStatusCode);
    assertEquals(expectedStatusCode, actualStatusCode);
}
```

- **Annotation**: @Given defines a Cucumber step for validating the status code.
- **Parameters**:
  - expectedStatusCode: The expected HTTP status code (e.g., 200, 201), passed from the feature file.
- **Steps**:
  1. Retrieve the actual status code from the response using response.getStatusCode().
  2. Write the actual status code to the Cucumber report.
  3. Use assertEquals to compare the expected and actual status codes.

---

8. Step Definition: Validate Element in Response

```
@Then("User validates {string} of {string} in response")
public void user_validates_of_in_response(String value, String element) {
    String nameValue = JsonPath.read(response.asString(), "$.result.name");
    scenario.write("name: " + nameValue);
    assertEquals(value, nameValue);
}
```

- **Annotation**: @Then defines a Cucumber step for validating a specific element in the response.
- **Parameters**:
  - value: The expected value of the element.
  - element: The name of the element to validate (e.g., name).
- **Steps**:
  1. Use **JsonPath** to extract the value of the element from the response.
  2. Write the extracted value to the Cucumber report.
  3. Use assertEquals to compare the expected and actual values.

---

9. Step Definition: Create Request Body for GIFT Expense

```
@Given("User creates GIFT request body {string},{string},{string},{string} for GIFT expense API")
public void user_creates_GIFT_request_body_for_GIFT_expense_API(String name, String amount, String
expenseDateTime, String giftRecipient) {
    // Create payload
    payload = createGIFTExpensePayload(name, amount, expenseDateTime, giftRecipient);
    // Write request data/payload to report
    scenario.write(payload);
    scenario.embed(payload.getBytes(), "application/json");
}
```

- **Annotation**: @Given defines a Cucumber step for creating a request payload for a **GIFT** expense.
- **Parameters**:
  - name, amount, expenseDateTime, giftRecipient: Values passed from the feature file to construct the payload.
- **Steps**:
  1. Call createGIFTExpensePayload (from ReusableMethods) to create the JSON payload.
  2. Write the payload to the Cucumber report.
  3. Embed the payload in the Cucumber report as JSON.

---

10. Step Definition: Create Request Body for TRAVEL Expense

```
@Given("User creates TRAVEL request body {string},{string},{string} for TRAVEL expense API")
public void user_creates_TRAVEL_request_body_for_TRAVEL_expense_API(String name, String amount,
String expenseDateTime) {
    // Create payload
    payload = createTRAVELExpensePayload(name, amount, expenseDateTime);
    // Write request data/payload to report
    scenario.write(payload);
    scenario.embed(payload.getBytes(), "application/json");
}
```

- **Annotation**: @Given defines a Cucumber step for creating a request payload for a **TRAVEL** expense.
- **Parameters**:
  - name, amount, expenseDateTime: Values passed from the feature file to construct the payload.
- **Steps**:
  1. Call createTRAVELExpensePayload (from ReusableMethods) to create the JSON payload.
  2. Write the payload to the Cucumber report.
  3. Embed the payload in the Cucumber report as JSON.

---

11. Step Definition: Validate Each Element in Response

```
@Given("User validate each {string} in response")
public void user_validate_each_in_response(String element) {
    String jsonpath = "$.result." + element;
    Object value = JsonPath.read(response.asString(), jsonpath);
    scenario.write(element + " value: " + value);
}
```

- **Annotation**: `@Given` defines a Cucumber step for validating a specific element in the response.
- **Parameters**:
  - `element`: The name of the element to validate (e.g., `amount`, `expenseType`).
- **Steps**:
  1. Construct the JSONPath expression to extract the element from the response.
  2. Use **JsonPath** to extract the value of the element.
  3. Write the extracted value to the Cucumber report.

Key Features of the Class

1. **Reusability**:
   - The class extends `ReusableMethods`, inheriting reusable methods for API testing.
2. **Modularity**:
   - Each step definition has a single responsibility, making the code easy to maintain and extend.
3. **Cucumber Reporting**:
   - The class uses `scenario.write()` and `scenario.embed()` to write logs and embed data (e.g., request/response) in the Cucumber report.
4. **JSON Handling**:
   - The class uses **JsonPath** to parse JSON responses and extract specific data.
5. **Assertions**:
   - The class uses `assertEquals` to validate responses (e.g., status codes, element values).

Example Feature File

Here’s an example of how the step definitions might be used in a Cucumber feature file:

```
Feature: Create Expense

  Scenario: Create a TRAVEL expense
    Given User creates TRAVEL request body "Flight", "500", "2023-10-01" for TRAVEL expense API
    When User submits POST request to "CreateExpense_URL" api
    Then User validates the status code is 201
    And User validates "Flight" of "name" in response

  Scenario: Create a GIFT expense
    Given User creates GIFT request body "Birthday Gift", "100", "2023-10-05", "John" for GIFT expense
API
    When User submits POST request to "CreateExpense_URL" api
    Then User validates the status code is 201
    And User validates "Birthday Gift" of "name" in response
```

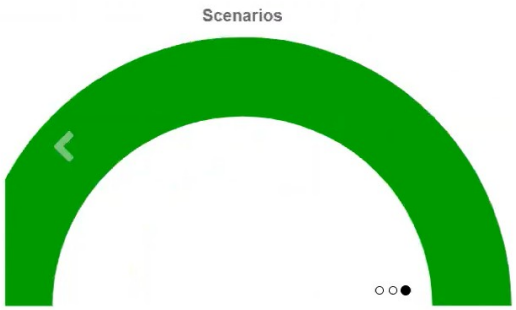
Conclusion

The CreateExpenseSteps class provides a structured and reusable way to test the creation of **GIFT**, **TRAVEL**, and **OTHER** expenses using **Cucumber** and **RestAssured**. It demonstrates how to:

- Create request payloads for different expense types.
- Submit POST requests and validate responses.
- Write logs and embed data in the Cucumber report.
- Use **JsonPath** to extract and validate specific elements in the response.

This approach is highly effective for API testing, especially in Behavior-Driven Development (BDD) frameworks like Cucumber.

Running multiple scenarios: @apiRegression on TestRunner



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Create GIFT Expense	3	0	0	0	0	3	1	0	1	3.453	Passed
Create Expense - Type Other	16	0	0	0	0	16	4	0	4	1.189	Passed
Create TRAVEL Expense	3	0	0	0	0	3	1	0	1	0.301	Passed
Create TRAVEL Expense	95	0	0	0	0	95	19	0	19	5.251	Passed
Dashboard API	3	0	0	0	0	3	1	0	1	0.795	Passed
Personal Info API	3	0	0	0	0	3	1	0	1	0.635	Passed
Personal Info API FLAG	88	0	0	0	0	88	22	0	22	0.212	Passed
Personal Info API	66	0	0	0	0	66	22	0	22	4.412	Passed
Token API	4	0	0	0	0	4	1	0	1	0.105	Passed
	281	**0**	**0**	**0**	**0**	**281**	**72**	**0**	**72**	**16.357**	**9**
	100.00%	**0.00%**	**0.00%**	**0.00%**	**0.00%**		**100.00%**	**0.00%**			**100.00%**

Last API homework: OMDB api -Interview AssignmentPostman collection:

OMDB.postman_collection.json


```
{
        "info": {
                "_postman_id": "ac9848c4-653a-4443-a9b0-c273f489d312",
                "name": "OMDB",
                "schema":
https://schema.getpostman.com/json/collection/v2.1.0/collection.json
```


Assignment details:

Stem QE Coding Exercise.txt

Stem QE Coding Exercise
*** Requirement && Steps to complete task
*** This part is assignment description.
 Import postman collection to understand these apis.

We want to test what is the best solution/program you can come up with when working with REST APIs. For this exercise, we are going to work with OMDb API.

Please see the documentation and get familiar with it. (I found using Postman to exercise this API particularly useful)

The assignment is as follows:

Write code that does the following
Goto https://www.omdbapi.com/ and sign up for an API Key. You'll need this API key to run the test.

Build a client that contains following methods using the OMDb API:

search(string): returns a list of all results that "Title" contains that string. Takes pagination in account when generating the list. Query uses "s" parameter as specified here.
get_by_id(string): returns the result based on "imdbID" matches with that input id (e.g. tt999999). Query uses "i" parameter as specified here.
get_by_title(string): returns the result based on input string where "Title" exactly macthes. Query uses "t" parameter as specified here.

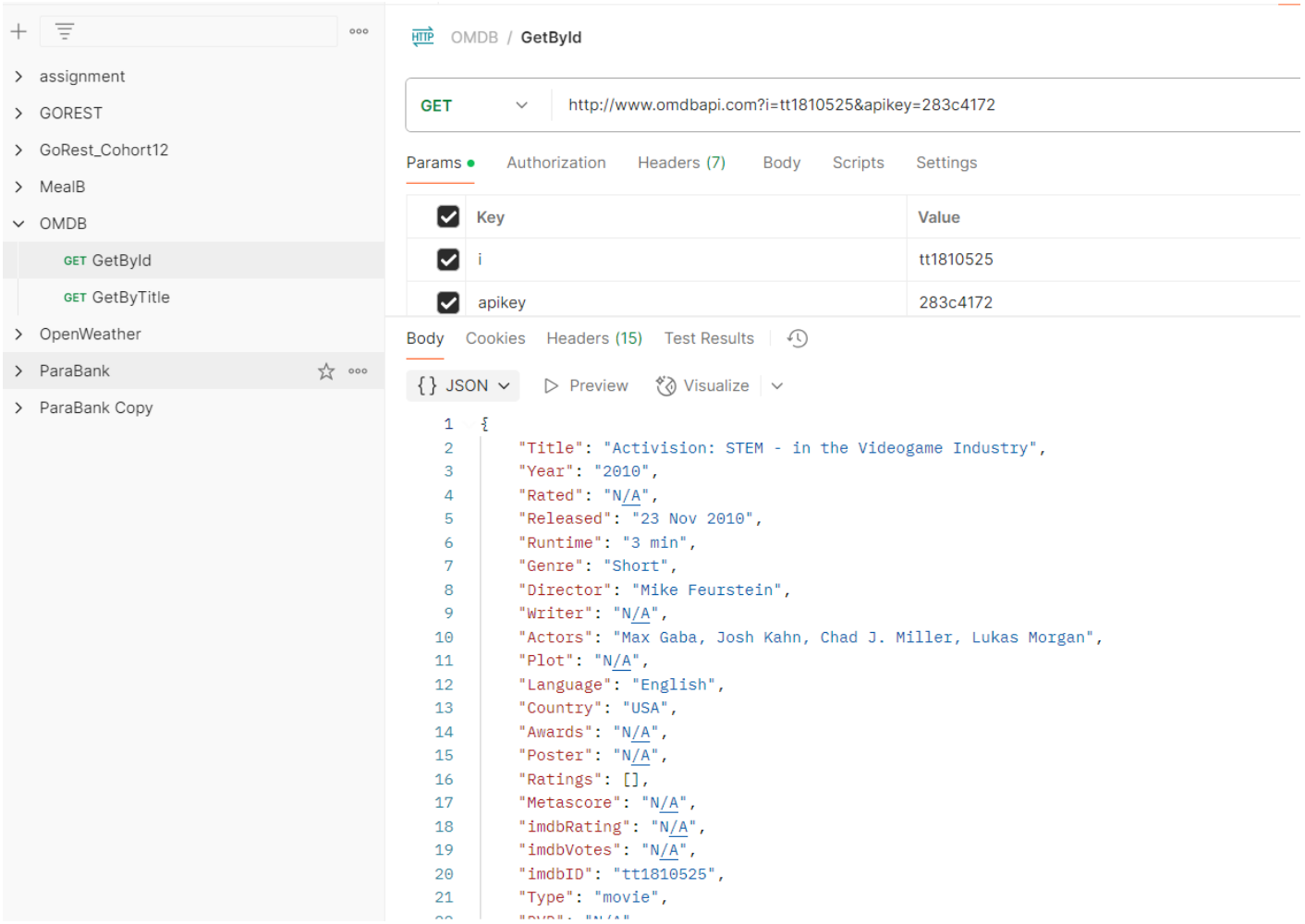
****** **Steps to complete task**

Write an automation code that does the following:
1. Using search method, search for all items that match the search string stem
2. Assert that the result should contain at least 30 items
3. Assert that the result contains items titled The STEM Journals and Activision: STEM - in the Videogame Industry
4. From the list returned by search above, get imdbID for item titled Activision: STEM - in the Videogame Industry and use it to get details on this movie using get_by_id method.
5. Assert that the movie was released on 23 Nov 2010 and was directed by Mike Feurstein
6. Using get_by_title method, get item by title "The STEM Journals" and assert that the Plot contains the string Science, Technology, Engineering and Math and has a "Runtime" of 22 minutes.

We will use 3 Get methods:

Get Method : GetById

`http://www.omdbapi.com?i=tt1810525&apikey=283c4172`

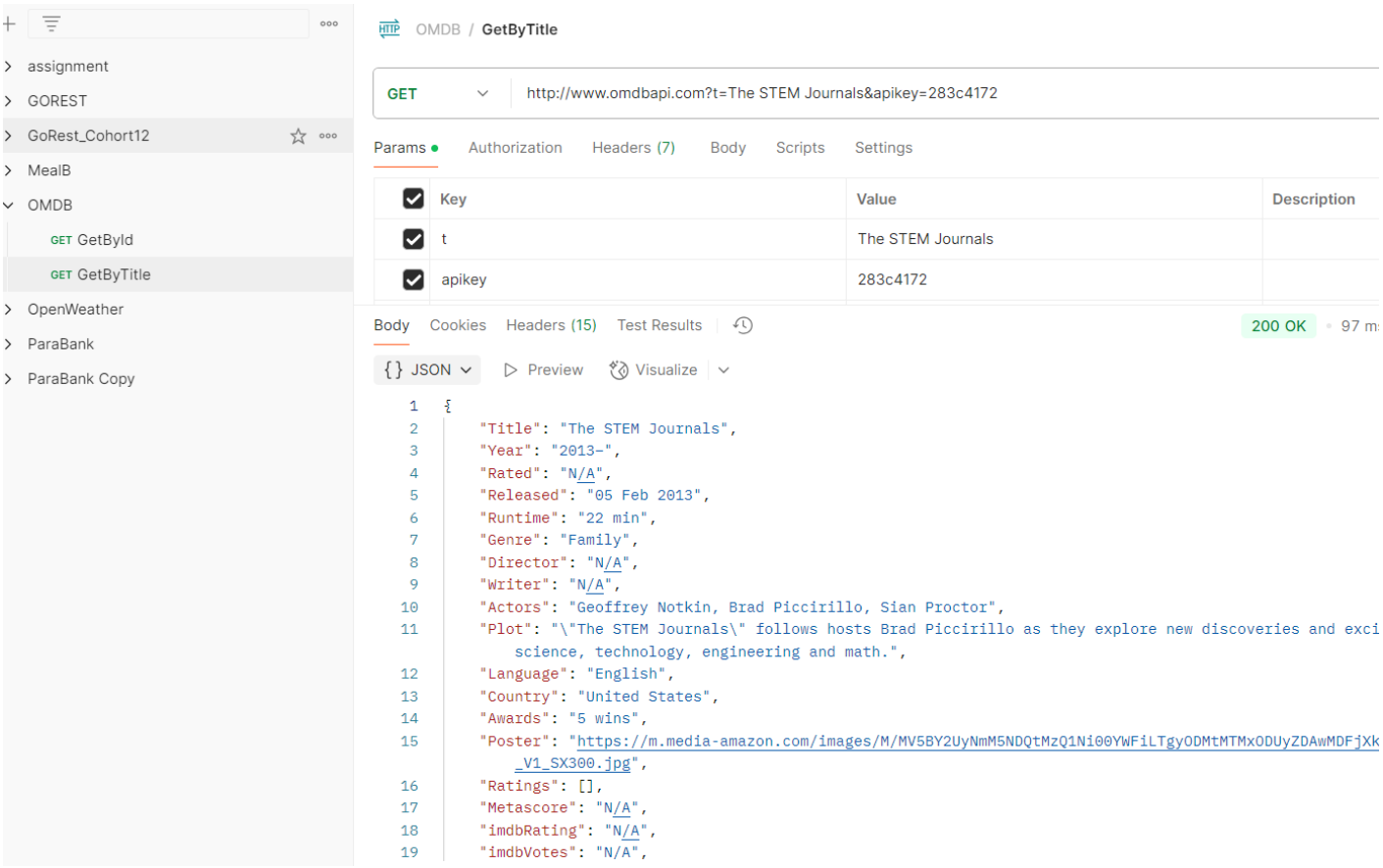



```
{
    "Title": "Activision: STEM - in the Videogame Industry",
    "Year": "2010",
    "Rated": "N/A",
    "Released": "23 Nov 2010",
    "Runtime": "3 min",
    "Genre": "Short",
    "Director": "Mike Feurstein",
    "Writer": "N/A",
    "Actors": "Max Gaba, Josh Kahn, Chad J. Miller, Lukas Morgan",
    "Plot": "N/A",
    "Language": "English",
    "Country": "USA",
    "Awards": "N/A",
    "Poster": "N/A",
    "Ratings": [],
    "Metascore": "N/A",
    "imdbRating": "N/A",
    "imdbVotes": "N/A",
    "imdbID": "tt1810525",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "N/A",
    "Production": "N/A",
```

```
    "Website": "N/A",
    "Response": "True"
}
```


Get Method : GetByTitle

http://www.omdbapi.com?t=The STEM Journals&apikey=283c4172

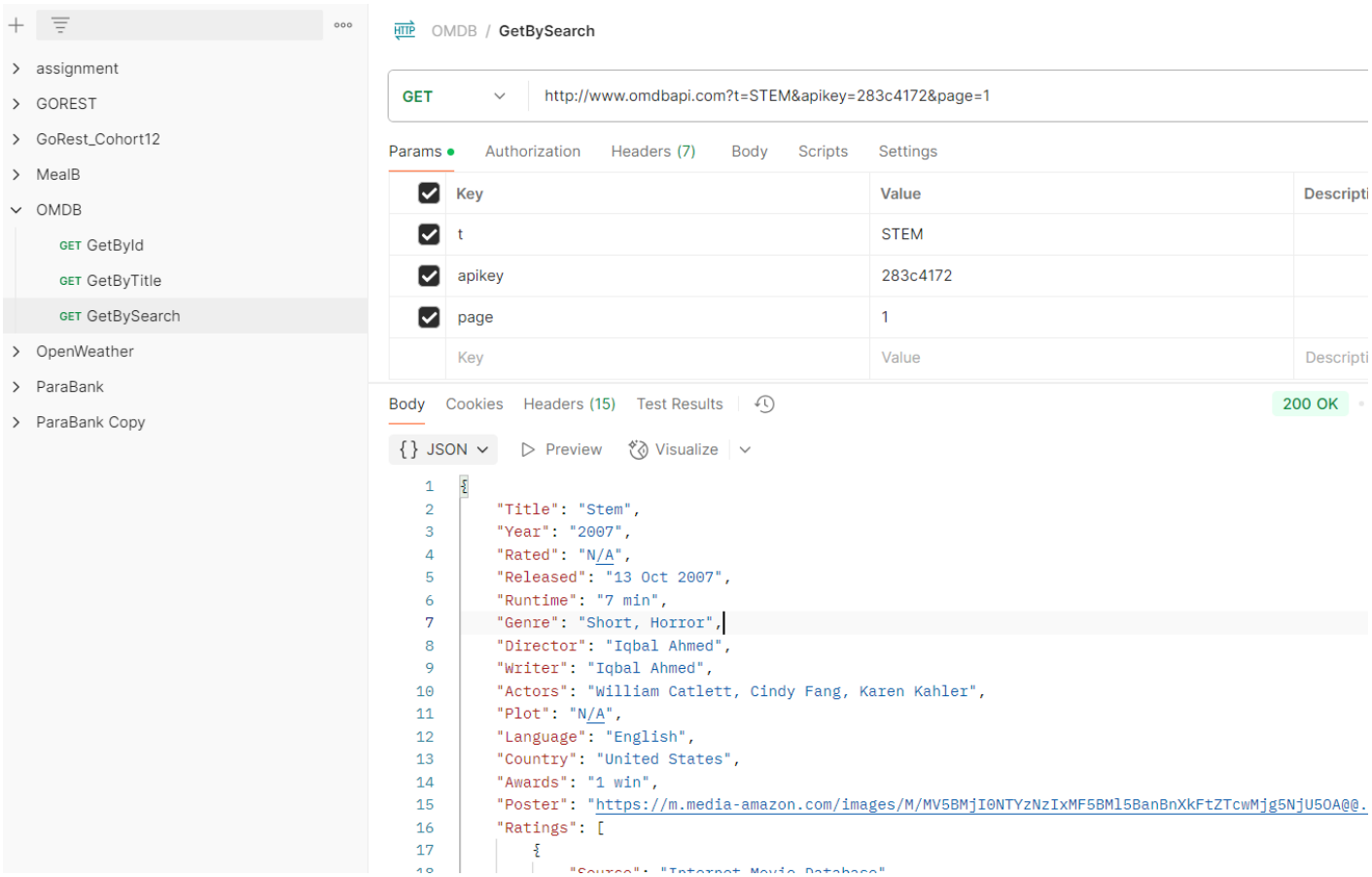



```
{
    "Title": "The STEM Journals",
    "Year": "2013–",
    "Rated": "N/A",
    "Released": "05 Feb 2013",
    "Runtime": "22 min",
    "Genre": "Family",
    "Director": "N/A",
    "Writer": "N/A",
    "Actors": "Geoffrey Notkin, Brad Piccirillo, Sian Proctor",
    "Plot": "\"The STEM Journals\" follows hosts Brad Piccirillo as they explore new discoveries and exciting careers in the fields of science, technology, engineering and math.",
    "Language": "English",
    "Country": "United States",
    "Awards": "5 wins",
```

```
    "Poster":
"https://m.media-amazon.com/images/M/MV5BY2UyNmM5NDQtMzQ1Ni00YWFiLTgyODMtMTMxODUyZDAwMDFjXkEyXkFqcGdeQXVyMTk1NDA2Mzc@._V1_SX300.jpg",
    "Ratings": [],
    "Metascore": "N/A",
    "imdbRating": "N/A",
    "imdbVotes": "N/A",
    "imdbID": "tt2717058",
    "Type": "series",
    "totalSeasons": "3",
    "Response": "True"
}
```


Get Method : GetBySearch

http://www.omdbapi.com?t=s&apikey=283c4172&page=1

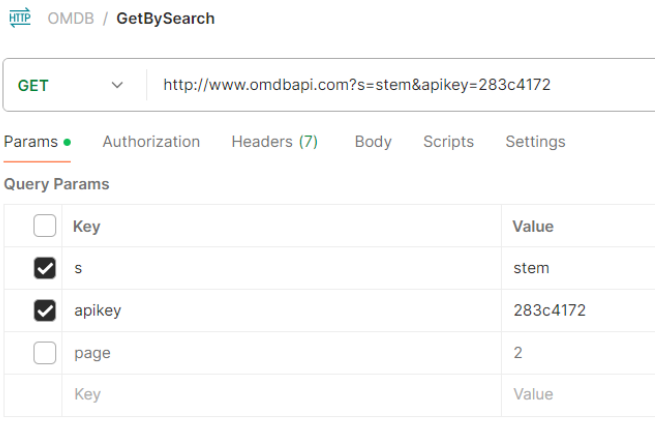


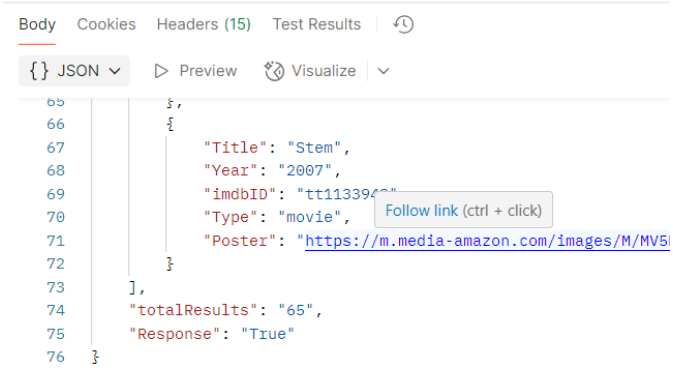

```
{
    "Title": "M*A*S*H",
    "Year": "1970",
    "Rated": "R",
    "Released": "18 Mar 1970",
    "Runtime": "116 min",
    "Genre": "Comedy, Drama, War",
    "Director": "Robert Altman",
    "Writer": "Richard Hooker, Ring Lardner Jr.",
    "Actors": "Donald Sutherland, Elliott Gould, Tom Skerritt",
```

```
    "Plot": "The staff of a Korean War field hospital uses humor and hijinks to keep their sanity in the face of the horror of war.",
    "Language": "English, Japanese, Korean, Latin",
    "Country": "United States",
    "Awards": "Won 1 Oscar. 15 wins & 26 nominations total",
    "Poster": "https://m.media-amazon.com/images/M/MV5BZDI0MDNiNWMtYzc5NS00MTc2LWJlMjktM2RkZTEyMzZjOTIwXkEyXkFqcGc@._V1_SX300.jpg",
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "7.3/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "86%"
        },
        {
            "Source": "Metacritic",
            "Value": "80/100"
        }
    ],
    "Metascore": "80",
    "imdbRating": "7.3",
    "imdbVotes": "79,556",
    "imdbID": "tt0066026",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$81,600,000",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
```


totalResults:65. If we divide it by 10>> 6 pages + one page

```
int totalPage=(totalResults/10)+1;
```





Create java class:

1. **OMDBmethods.**
2. **TestHomework extended from OMDBmethods.**

```
package omdbHomework;

import io.restassured.response.Response;
import static io.restassured.RestAssured.given;
public class OMDBmethods {

     String url="http://www.omdbapi.com/";
     String apiKey="283c4172";

     public String search(String text, int page) {
          Response resp=given()
                              .param("s", text)
                              .param("apikey", apiKey)
                              .param("page",page)
                         .when()
                              .get(url);
           return resp.prettyPrint();
     }
```

```
     //getByID(Sting): returns the result based on "imdbID" matches with
that input id(e.g. tt999999)
     // Query uses "i" parameters as specified here

     public String getByID(String id) {
          Response resp=given()
                              .param("i", id)
                              .param("apikey", apiKey)
                         .when()
                              .get(url);
           return resp.prettyPrint();
     }


     //getByID(Sting): returns the result based on input string where
"Title" exactly matches
     // Query uses "t" parameters as specified here


     public String getByTitle(String title) {
          Response resp=given()
                              .param("t", title)
                              .param("apikey", apiKey)
                         .when()
                              .get(url);
           return resp.prettyPrint();
     }
}
```

Explaination of OMDBmethods:

The provided code is a **Java class** named OMDBmethods that interacts with the **OMDb API** (Open Movie Database API). This class contains methods to perform various operations like searching for movies, fetching movie details by IMDb ID, and fetching movie details by title. Below is a detailed explanation of the code, broken down into sections:

---

1. Package and Imports

```
package omdbHomework;

import io.restassured.response.Response;
import static io.restassured.RestAssured.given;
```

- **Package**: The class belongs to the omdbHomework package.
- **Imports**:
    - **io.restassured.response.Response**: Used to handle the response from the API.

- **static io.restassured.RestAssured.given**: Used to send HTTP requests using RestAssured.

---

2. Class Definition

```
public class OMDBmethods {
```

- **Class**: OMDBmethods is a utility class that contains methods to interact with the OMDb API.

---

3. Class Variables

```
String url = "http://www.omdbapi.com/";
String apiKey = "283c4172";
```

- **Variables**:
  - url: The base URL for the OMDb API.
  - apiKey: The API key required to access the OMDb API. This key is obtained by signing up on the OMDb website.

---

4. Method: search

```
public String search(String text, int page) {
   Response resp = given()
                .param("s", text)
                .param("apikey", apiKey)
                .param("page", page)
              .when()
                .get(url);
   return resp.prettyPrint();
}
```

- **Purpose**: Searches for movies containing the specified text.
- **Parameters**:
  - text: The search string (e.g., "stem").
  - page: The page number for pagination (used to handle multiple pages of results).
- **Steps**:
  1. Use **RestAssured** to send a **GET request** to the OMDb API.
  2. Add query parameters:
     - s: The search string.
     - apikey: The API key.
     - page: The page number.
  3. Send the request to the base URL (http://www.omdbapi.com/).
  4. Return the response as a pretty-printed JSON string using resp.prettyPrint().

---

5. Method: getByID

```
public String getByID(String id) {
   Response resp = given()
```

```
                .param("i", id)
                .param("apikey", apiKey)
              .when()
                .get(url);
    return resp.prettyPrint();
}
```

- **Purpose**: Fetches movie details by IMDb ID.
- **Parameters**:
  - id: The IMDb ID of the movie (e.g., "tt1810525").
- **Steps**:
  1. Use **RestAssured** to send a **GET request** to the OMDb API.
  2. Add query parameters:
     - i: The IMDb ID.
     - apikey: The API key.
  3. Send the request to the base URL (http://www.omdbapi.com/).
  4. Return the response as a pretty-printed JSON string using resp.prettyPrint().

---

6. Method: getByTitle

```
public String getByTitle(String title) {
    Response resp = given()
                .param("t", title)
                .param("apikey", apiKey)
              .when()
                .get(url);
    return resp.prettyPrint();
}
```

- **Purpose**: Fetches movie details by title.
- **Parameters**:
  - title: The exact title of the movie (e.g., "The STEM Journals").
- **Steps**:
  1. Use **RestAssured** to send a **GET request** to the OMDb API.
  2. Add query parameters:
     - t: The movie title.
     - apikey: The API key.
  3. Send the request to the base URL (http://www.omdbapi.com/).
  4. Return the response as a pretty-printed JSON string using resp.prettyPrint().

---

7. Key Features of the Code

1. **RestAssured**:
   - The code uses **RestAssured**, a popular Java library for testing RESTful APIs, to send HTTP requests and handle responses.
2. **Query Parameters**:
   - The methods use query parameters (s, i, t, apikey) to interact with the OMDb API.
3. **Pagination**:
   - The search method supports pagination by accepting a page parameter.

4. **Pretty-Printed JSON**:
    - The responses are returned as pretty-printed JSON strings, making them easier to read and debug.

---

8. Example Usage

Here's how the `OMDBmethods` class can be used in a test or application:

Search for Movies

```
OMDBmethods omdb = new OMDBmethods();
String searchResponse = omdb.search("stem", 1);
System.out.println(searchResponse);
```

- **Output**: A JSON string containing the search results for movies with "stem" in their title.

Fetch Movie Details by IMDb ID

```
String movieDetails = omdb.getByID("tt1810525");
System.out.println(movieDetails);
```

- **Output**: A JSON string containing detailed information about the movie with IMDb ID "tt1810525".

Fetch Movie Details by Title

```
String movieDetails = omdb.getByTitle("The STEM Journals");
System.out.println(movieDetails);
```

- **Output**: A JSON string containing detailed information about the movie titled "The STEM Journals".

---

9. Example API Responses

Search Response


```
{
  "Search": [
    {
      "Title": "The STEM Journals",
      "Year": "2013–",
      "imdbID": "tt2717058",
      "Type": "series",
      "Poster":
"https://m.media-amazon.com/images/M/MV5BY2UyNmM5NDQtMzQ1Ni00YWFiLTgyODMtMTMxODUyZDAwMDFjXkEyXkFqcGdeQXVyMTk1NDA2Mzc@._V1_SX300.jpg"
    },
    {
      "Title": "Activision: STEM - in the Videogame Industry",
      "Year": "2010",
      "imdbID": "tt1810525",
      "Type": "movie",
      "Poster": "N/A"
    }
```

```
  ],
  "totalResults": "32",
  "Response": "True"
}
```


Movie Details Response


```
{
  "Title": "Activision: STEM - in the Videogame Industry",
  "Year": "2010",
  "Rated": "N/A",
  "Released": "23 Nov 2010",
  "Runtime": "3 min",
  "Genre": "Short",
  "Director": "Mike Feurstein",
  "Writer": "N/A",
  "Actors": "Max Gaba, Josh Kahn, Chad J. Miller, Lukas Morgan",
  "Plot": "N/A",
  "Language": "English",
  "Country": "USA",
  "Awards": "N/A",
  "Poster": "N/A",
  "Ratings": [],
  "Metascore": "N/A",
  "imdbRating": "N/A",
  "imdbVotes": "N/A",
  "imdbID": "tt1810525",
  "Type": "movie",
  "DVD": "N/A",
  "BoxOffice": "N/A",
  "Production": "N/A",
  "Website": "N/A",
  "Response": "True"
}
```


10. Conclusion

The `OMDBmethods` class provides a simple and reusable way to interact with the **OMDb API**. It includes methods for:

- Searching for movies by title.
- Fetching movie details by IMDb ID.
- Fetching movie details by title.

This class is highly modular and can be integrated into larger test automation frameworks or applications. By using **RestAssured**, the code is concise, readable, and easy to maintain.

TestHomework class extended from OMDBmethods:

```
package omdbHomework;

import static org.junit.Assert.assertEquals;
```

```java
import static org.junit.Assert.assertTrue;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.junit.Test;

import com.jayway.jsonpath.JsonPath;

public class TestHomework extends OMDBmethods{
	String response;
	String imdbID;
	List<String>titleList=new ArrayList<String>();
	List<Object>resultList=new ArrayList<Object>();

	@Test
	public void homework() {
		//1. Using search method, search for all items that match the search string "stem"
		  response=search("stem", 1);
		//2. Assert that the result should contain at least 30 items
		  String totalnum=JsonPath.read(response, "$.totalResults");
		  System.out.println(totalnum);
		  int totalResults=Integer.valueOf(totalnum);
		  assertTrue(totalResults>30);

		  //3. Assert that the result contains items titled
		//   The STEM Journals and Activision: STEM - in the Videogame Industry
		  int totalPage=(totalResults/10)+1;
		 List<String> titleListTotal=new ArrayList<>();
		 List<String> resultListTotal=new ArrayList<>();

		 for (int i = 1; i <=totalPage; i++) {

			response =search("stem", i);
			List<String>titleList=JsonPath.read(response, "$.Search[*].Title");
			titleListTotal.addAll(titleList);

			//titleList.addAll((Collection<? extends String>) JsonPath.read(response, "$.Search[*].Title"));

			List<String>resultList=JsonPath.read(response, "$.Search[*]");
			resultListTotal.addAll(resultList);

			//resultList.addAll((Collection<? extends String>) JsonPath.read(response, "$.Search[*]"));
```

```
        }
        //3. Assert that the result contains items titled The STEM
Journals and Activision: STEM - in the Videogame Industry

        assertTrue(titleList.contains("The STEM Journals"));
        assertTrue(titleList.contains("Activision: STEM - in the
Videogame Industry"));


        //4. From the list returned by search above,
        //get imdbID for item titled Activision: STEM - in the
Videogame Industry

        for(Object eachResult: resultList) {
            if(JsonPath.read(eachResult,
"$.Title").equals("Activision: STEM - in the Videogame Industry")) {
                imdbID=JsonPath.read(eachResult, "$.imdbID");
            }
        }

        //and use it to get details on this movie using get_by_id
method.
        response =getByID(imdbID);

        //5. Assert that the movie was released on 23 Nov 2010 and was
directed by Mike Feurstein
        String released=JsonPath.read(response, "$.Released");
        String directed=JsonPath.read(response, "$.Director");

        System.out.println(released);
        System.out.println(directed);

        assertEquals(released, "23 Nov 2010");
        assertEquals(directed, "Mike Feurstein");

    }

}
```

Explaination of TestHomework Class:

The TestHomework class is a JUnit test class that extends the OMDBmethods class. It tests the functionality of the OMDB API by performing a series of steps, including searching for movies, validating search results, and verifying specific details about a movie. Below is a **detailed explanation** of the code:

---

1. Class Declaration and Imports

```
package omdbHomework;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.junit.Test;

import com.jayway.jsonpath.JsonPath;

public class TestHomework extends OMDBmethods {
```

- **Purpose**: Declares the TestHomework class, which extends OMDBmethods.
- **Imports**:
  - org.junit.*: For JUnit testing annotations and assertions.
  - java.util.*: For working with collections like List and ArrayList.
  - com.jayway.jsonpath.JsonPath: For extracting data from JSON responses.

---

2. Class Variables

```
String response;
String imdbID;
List<String> titleList = new ArrayList<String>();
List<Object> resultList = new ArrayList<Object>();
```

- **Purpose**: Declares variables to store:
  - response: The JSON response from the API.
  - imdbID: The IMDb ID of a specific movie.
  - titleList: A list to store movie titles.
  - resultList: A list to store full movie details.

---

3. Test Method Declaration

```
@Test
public void homework() {
```

- **Purpose**: Declares a test method named homework. The @Test annotation marks it as a JUnit test case.

---

4. Step 1: Search for Movies

```
response = search("stem", 1);
```

- **Purpose**: Calls the search method (inherited from OMDBmethods) to search for movies containing the keyword "stem".
- **Details**:
  - The search method sends a request to the OMDB API and returns the JSON response.
  - The response is stored in the response variable.

---

5. Step 2: Assert Total Results

```
String totalnum = JsonPath.read(response, "$.totalResults");
System.out.println(totalnum);
int totalResults = Integer.valueOf(totalnum);
assertTrue(totalResults > 30);
```

- **Purpose**: Validates that the search returns at least 30 items.
- **Details**:
  - JsonPath.read(response, "$.totalResults"): Extracts the totalResults field from the JSON response.
  - Integer.valueOf(totalnum): Converts the totalResults string to an integer.
  - assertTrue(totalResults > 30): Asserts that the total number of results is greater than 30.

---

6. Step 3: Iterate Through All Pages

```
int totalPage = (totalResults / 10) + 1;
List<String> titleListTotal = new ArrayList<>();
List<String> resultListTotal = new ArrayList<>();

for (int i = 1; i <= totalPage; i++) {
    response = search("stem", i);
    List<String> titleList = JsonPath.read(response, "$.Search[*].Title");
    titleListTotal.addAll(titleList);

    List<String> resultList = JsonPath.read(response, "$.Search[*]");
    resultListTotal.addAll(resultList);
}
```

- **Purpose**: Iterates through all pages of search results to collect movie titles and details.
- **Details**:
  - totalPage = (totalResults / 10) + 1: Calculates the total number of pages (assuming 10 results per page).
  - The for loop iterates through each page:
    - Calls the search method for the current page.
    - Extracts movie titles using JsonPath.read(response, "$.Search[*].Title").
    - Extracts full movie details using JsonPath.read(response, "$.Search[*]").
    - Adds the results to titleListTotal and resultListTotal.

---

7. Step 4: Validate Specific Titles

```
assertTrue(titleList.contains("The STEM Journals"));
assertTrue(titleList.contains("Activision: STEM - in the Videogame Industry"));
```

- **Purpose**: Validates that the search results contain specific movie titles.
- **Details**:
  - assertTrue(titleList.contains(...)): Asserts that titleList contains the specified titles.

---

8. Step 5: Extract IMDb ID

```
for (Object eachResult : resultList) {
    if (JsonPath.read(eachResult, "$.Title").equals("Activision: STEM - in the Videogame Industry")) {
        imdbID = JsonPath.read(eachResult, "$.imdbID");
    }
}
```

- **Purpose**: Extracts the IMDb ID for the movie titled "Activision: STEM - in the Videogame Industry".
- **Details**:
    - Iterates through resultList to find the movie with the specified title.
    - Uses JsonPath.read(eachResult, "$.imdbID") to extract the IMDb ID.

---

9. Step 6: Get Movie Details

```
response = getByID(imdbID);
```

- **Purpose**: Calls the getByID method (inherited from OMDBmethods) to fetch details for the movie with the extracted IMDb ID.
- **Details**:
    - The getByID method sends a request to the OMDB API and returns the JSON response.
    - The response is stored in the response variable.

---

10. Step 7: Validate Movie Details

```
String released = JsonPath.read(response, "$.Released");
String directed = JsonPath.read(response, "$.Director");

System.out.println(released);
System.out.println(directed);

assertEquals(released, "23 Nov 2010");
assertEquals(directed, "Mike Feurstein");
```

- **Purpose**: Validates the release date and director of the movie.
- **Details**:
    - JsonPath.read(response, "$.Released"): Extracts the Released field from the JSON response.
    - JsonPath.read(response, "$.Director"): Extracts the Director field from the JSON response.
    - assertEquals(released, "23 Nov 2010"): Asserts that the release date is "23 Nov 2010".
    - assertEquals(directed, "Mike Feurstein"): Asserts that the director is "Mike Feurstein".

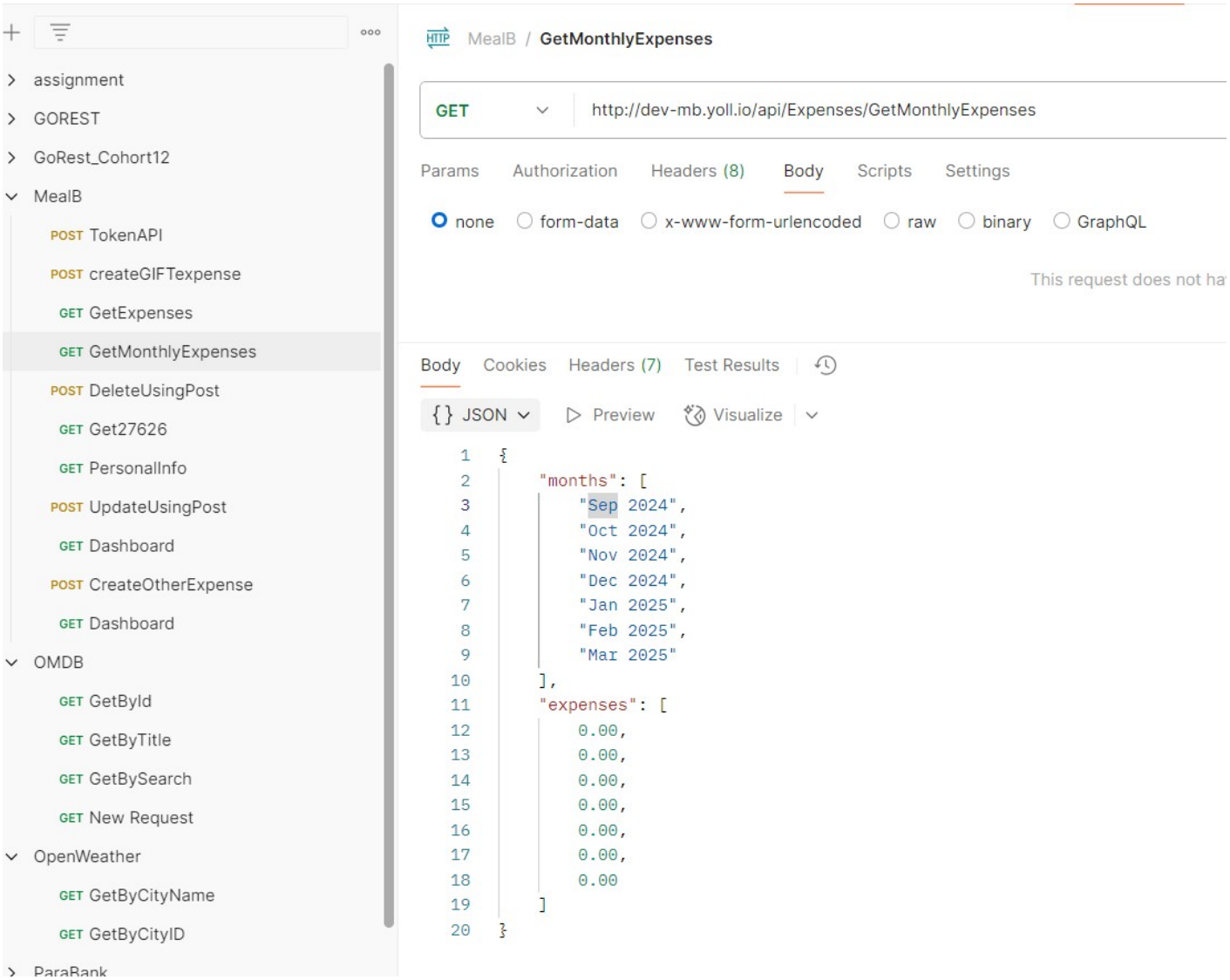
Summary of the Test

1. **Search for Movies**:
    - Searches for movies containing the keyword "stem".
    - Validates that the search returns at least 30 results.
2. **Iterate Through Pages**:

    - Collects all movie titles and details by iterating through all pages of search results.
3. **Validate Specific Titles**:
    - Asserts that the search results contain specific movie titles.
4. **Extract IMDb ID**:
    - Extracts the IMDb ID for a specific movie.
5. **Get Movie Details**:
    - Fetches details for the movie using its IMDb ID.
6. **Validate Movie Details**:
    - Asserts the release date and director of the movie.

Key Points

- The test demonstrates how to interact with the OMDB API using methods inherited from `OMDBmethods`.
- It uses **JSONPath** to extract data from JSON responses.
- It validates search results and specific movie details using JUnit assertions.
- The test ensures that the API is functioning as expected by checking both search results and individual movie details.




```
Feature: Get Monthly Expenses

@monthlyExpense
Scenario: Validate Get Monthly Expenses shows 7 months

     Given User sends GET request to "GetMonthlyExpenses_URL" api
     When User validate status code is 200
     Then User validates results shows past seven months
```

TestRunner Class:

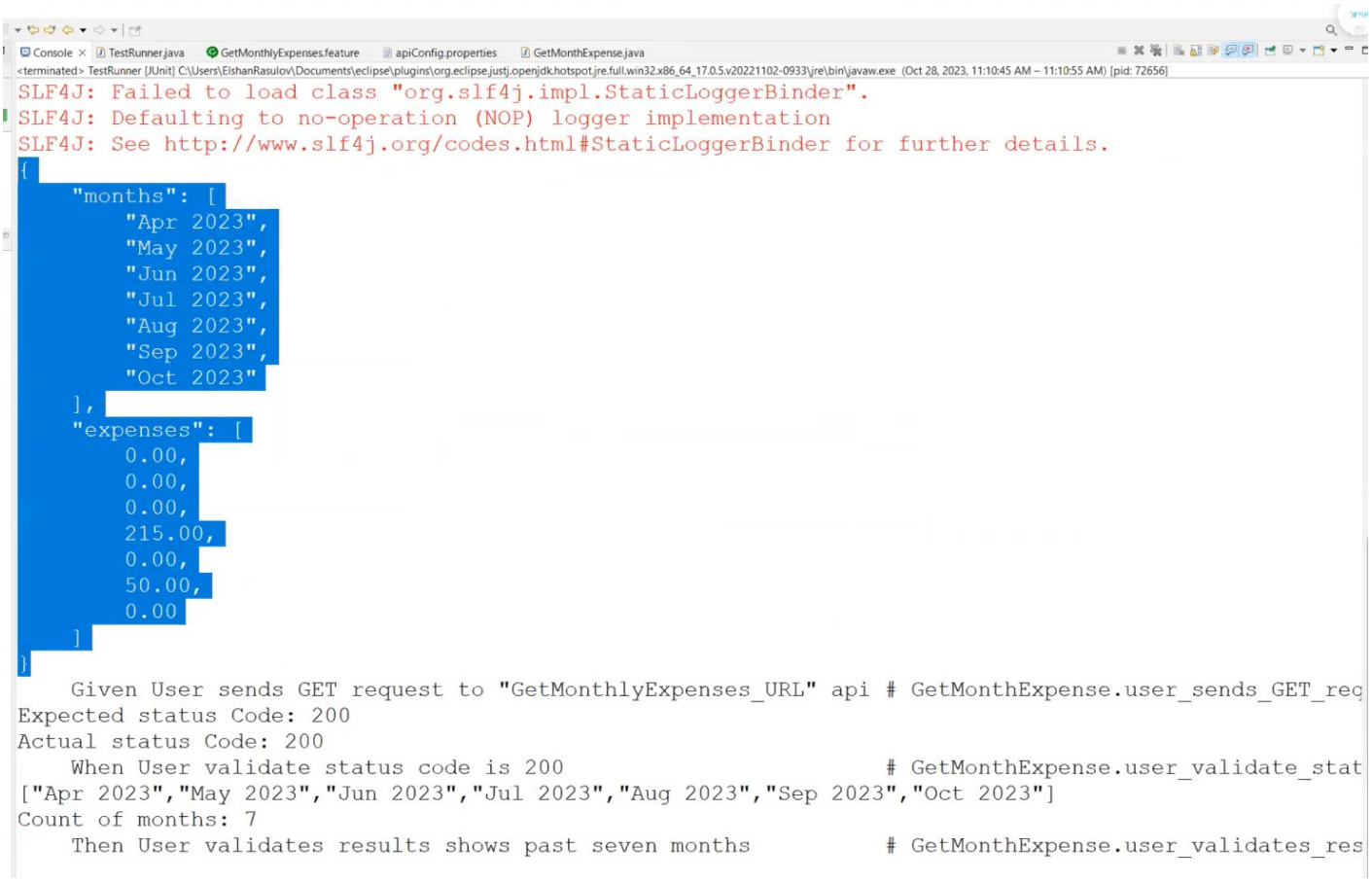
```
package runner;


import org.junit.runner.RunWith;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
```

```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "./src/test/resources/features",
        glue={"stepDefinitions"},
        dryRun = false,
        monochrome = true,
        tags = { "@monthlyExpense" },
        plugin = { "pretty", "html:target/cucumber-html-report",
"json:target/cucumber.json" }
)


public class TestRunner {

}
```

GetMonthExpense extended from ReusableMethods

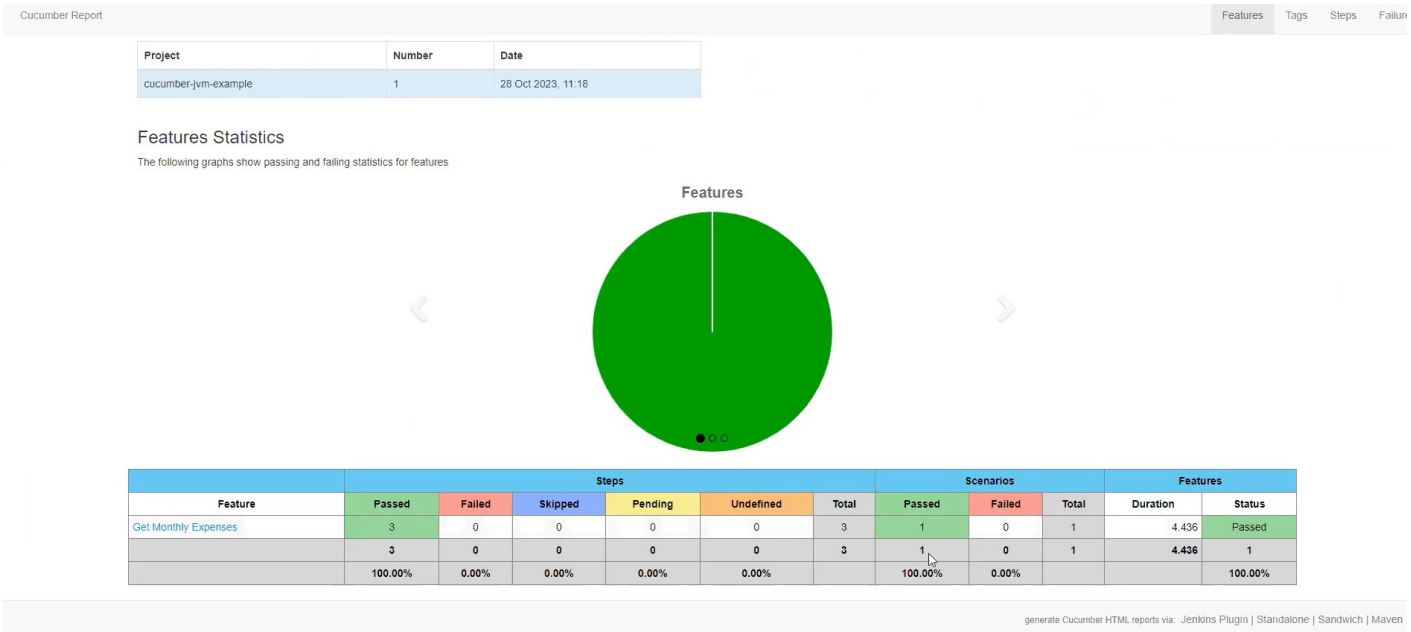
```
package stepDefinitions;
import static io.restassured.RestAssured.given;
import static org.junit.Assert.assertEquals;

import java.util.List;

import com.jayway.jsonpath.JsonPath;

import io.cucumber.core.api.Scenario;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import io.restassured.response.Response;
import utils.ReusableMethods;

public class GetMonthExpense extends ReusableMethods {
     Response response;
     Scenario scenario;


     @Before
     public void keepScenario(Scenario scenario){
          this.scenario=scenario;
     }

     @Given("User sends GET request to {string} api")
```

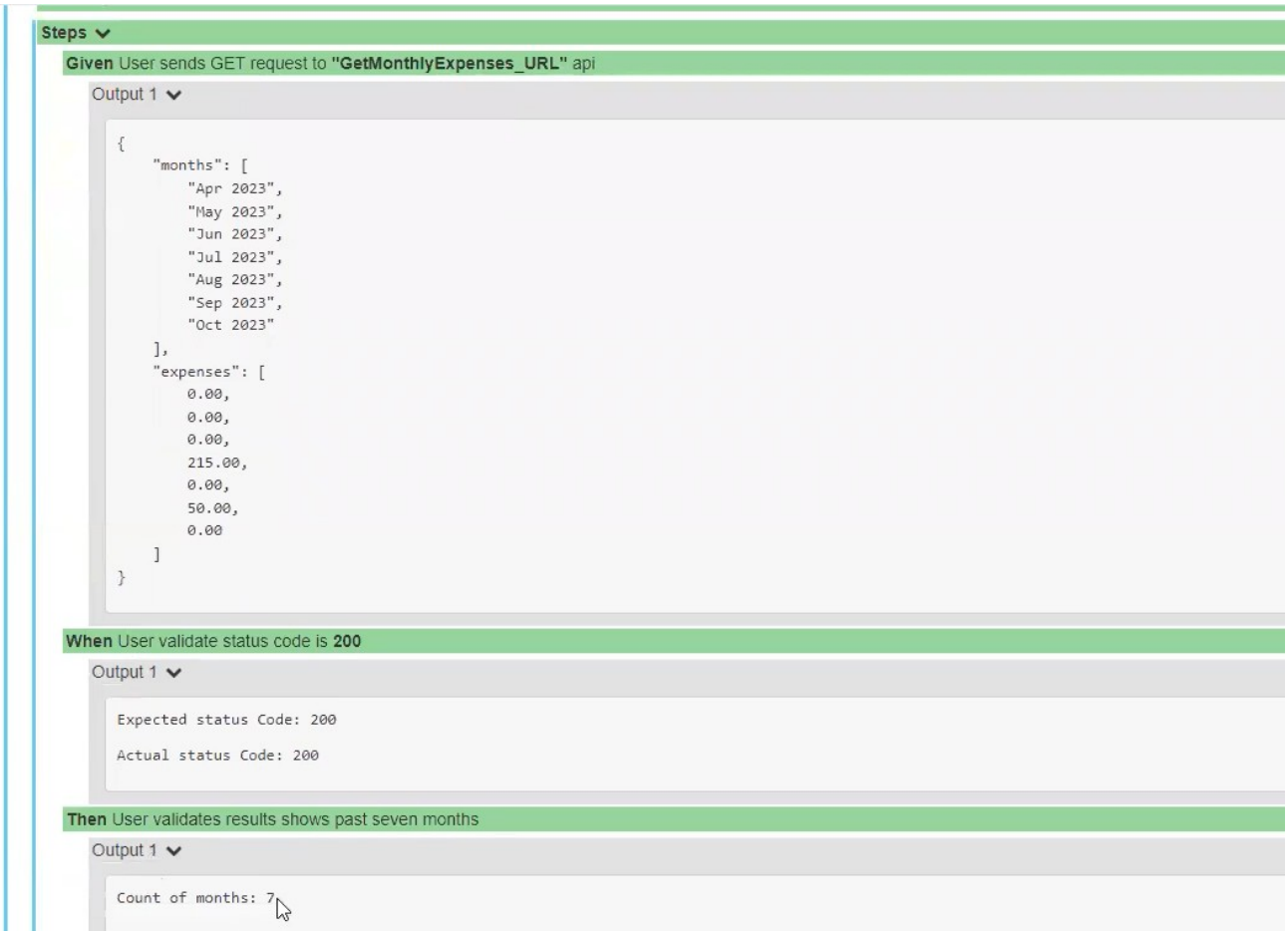
```
	public void user_sends_GET_request_to_api(String url) {

			response=given()
					.header("Authorization", "Bearer
"+getToken())
				.when()
				  .get(property.getProperty(url));
			response.prettyPrint();
			scenario.write(response.asPrettyString());

	}

	@When("User validate status code is {int}")
	public void user_validate_status_code_is(int statusCode) {

		scenario.write("Expected status Code: "+statusCode);
		scenario.write("Actual status Code:
"+response.getStatusCode());

		assertEquals(statusCode,response.getStatusCode() );


	}

	@Then("User validates results shows past seven months")
	public void user_validates_results_shows_past_seven_months() {



List<String>months=JsonPath.read(response.asString(),"$.months[*]");
		scenario.write("Count of months: "+months.size());
		assertEquals(7, months.size());

	}

}
```



Report of @monthlyExpense:

Run it with maven !
Refresh the project





ReusableMethods class extended from Base:

```
package utils;

import static io.restassured.RestAssured.given;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.jayway.jsonpath.JsonPath;

import io.restassured.http.ContentType;
import io.restassured.response.Response;
import payloadBuilder.GiftExpenseRequestBuilder;
import payloadBuilder.OtherExpenseRequestBuilder;
import payloadBuilder.TokenRequestBuilder;
import payloadBuilder.TravelExpenseRequestBuilder;

public class ReusableMethods extends Base{
	/**
	 * @author ElshanRasulov
	 * @return Response
	 * @see This method submit POST request to delete expenses
	 */
```

```
    public Response submitDeleteExpenseRequest(Integer id) {
        Response response=given()
                                    .header("Authorization", "Bearer
"+getToken())
                                .when()
                                    .post(property.getProperty("Delete_UR
L")+id);
        return response;
    }

    /**
     * @author ElshanRasulov
     * @return String
     * @see This method create payload for TRAVEL type expense and
convert it to String
     */

    public String createTRAVELExpensePayload(String name,String amount,
String expenseDateTime){
        TravelExpenseRequestBuilder payload=new
TravelExpenseRequestBuilder();
            payload.setName(name);
            payload.setAmount(amount);
            payload.setExpenseDateTime(expenseDateTime);
            payload.setExpenseType("Travel");

        return convertObjectToString(payload);
    }


    /**
     * @author ElshanRasulov
     * @return String
     * @see This method create payload for GIFT type expense and
convert it to String
     */

    public String createGIFTExpensePayload(String name,String amount,
String expenseDateTime, String giftRecipient) {

        GiftExpenseRequestBuilder payload=new
GiftExpenseRequestBuilder();
        payload.setName(name);
        payload.setAmount(amount);
        payload.setExpenseDateTime(expenseDateTime);
        payload.setExpenseType("Gift");
        payload.setGiftRecipient(giftRecipient);

    return convertObjectToString(payload);
```

```
    }


    /**
     * @author ElshanRasulov
     * @return String
     * @see This method submits POST request to create expense
     */

    public Response submitCreateExpenseRequest(String payload) {

        Response response=given()
                                .header("Authorization", "Bearer
"+getToken())
                                .body(payload)
                                .contentType(ContentType.JSON)
                            .when()
                                .post(property.getProperty("CreateExp
ense_URL"));
        response.prettyPrint();

        return response;
    }




    /**
     * @author ElshanRasulov
     * @return String
     * @see This method create payload for OTHER type expense and
convert it to String
     */

    public String createOtherExpensePayload(String name,String amount,
String expenseDateTime,String expenseType, String otherExpenseNameId) {

        OtherExpenseRequestBuilder payload=new
OtherExpenseRequestBuilder();
        payload.setName(name);
        payload.setAmount(amount);
        payload.setExpenseType(expenseType);
        payload.setExpenseDateTime(expenseDateTime);
        payload.setOtherExpenseNameId(otherExpenseNameId);

      return convertObjectToString(payload);

    }
```

```
     /**
      * @author ElshanRasulov
      * @return String
      * @see This method submits POST request to generate new token
      */
     public String getToken() {

          Response tokenResp=given()
                                    .contentType(ContentType.JSON)
                                    .body(createTokenPayload())
                                .when()

     .post(property.getProperty("TOKEN_URL"));
          tokenResp.prettyPrint();

          String token=JsonPath.read(tokenResp.asString(),
"$.result.accessToken");

          return token;

     }



     public String createTokenPayload() {

      TokenRequestBuilder tokenReqBuilder=new TokenRequestBuilder();


tokenReqBuilder.setUsernameOrEmailAddress(property.getProperty("userID")
);
          tokenReqBuilder.setPassword(property.getProperty("password"));

          return convertObjectToString(tokenReqBuilder);
     }




     public String convertObjectToString(Object object) {
          String strPayload=null;
          try {
               ObjectMapper objMapper=new ObjectMapper();
                strPayload =
objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(object);
          } catch (JsonProcessingException e) {
               e.printStackTrace();
          }
```

```
      return strPayload;
      }


}
```

Base Class :

```
public static Properties property;

      static{
            try {

                property=new Properties();
                String
path="src/test/resources/configs/apiConfig.properties";
                File file=new File(path);
                FileInputStream fis=new FileInputStream(file);
                property.load(fis);

            }catch(Exception e) {
                 e.printStackTrace();
            }

      }


}
```